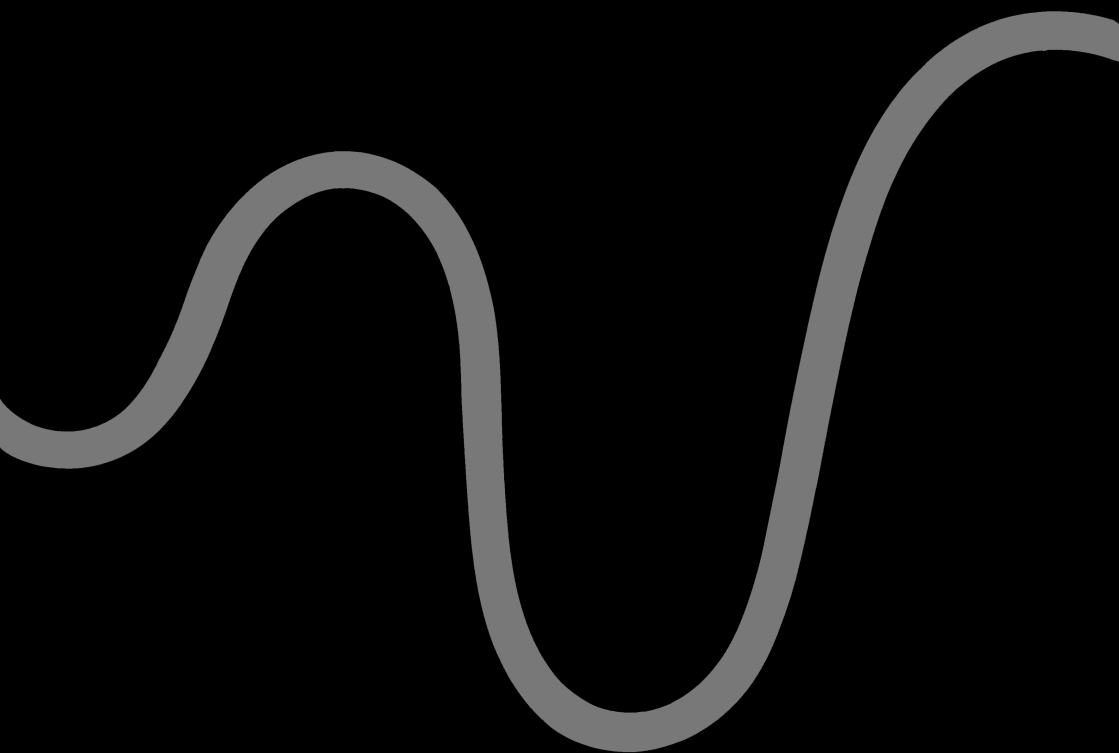
LOGICA E COMPUTER

MORTON L. SCHAGRIN
WILLIAM J. RAPAPORT
RANDALL R. DIPERT

McGraw-Hill

LOGICA E COMPUTER

Sezione di Intelligenza Artificiale
coordinata da Marco Colombetti

Indice

Prefazione

Questo è un libro di logica e, nello stesso tempo, un testo di programmazione e di Intelligenza Artificiale. L'accoppiamento non deve stupire: la logica è parte integrante dell'Intelligenza Artificiale da quando questa è nata, nel 1956. Uno dei primi sistemi "intelligenti", il Logic Theorist di Newell, Shaw e Simon, aveva il compito di costruire automaticamente le dimostrazioni di semplici teoremi tratti dai *Principia Mathematica* di Whitehead e Russell. In seguito, programmi per la dimostrazione di teoremi sono stati utilizzati per risolvere automaticamente problemi nei campi più disparati.
Con l'andare degli anni, il rapporto fra la logica matematica e l'Intelligenza Artificiale si è fatto complesso. In modo un po' approssimativo, si può dire oggi che la logica fornisce all'Intelligenza Artificiale le basi concettuali per formulare teorie precise di certe prestazioni intelligenti. Queste teorie danno poi la possibilità di valutare la correttezza e la completezza dei programmi costruiti per riprodurre le stesse prestazioni sul calcolatore.
In questo modo, la logica assume per l'Intelligenza Artificiale un ruolo simile a quello che l'analisi matematica ha verso la fisica. Anzi, l'analogia si può spingere più oltre. Se la fisica è debitrice all'analisi dal punto di vista del metodo, l'opposto è vero dal punto di vista della didattica: è attraverso gli esempi della meccanica, dell'acustica e così via che si possono rendere vividi e comprensibili i concetti di funzione, di derivata, d'integrale, di equazione differenziale. Schagrin, Rapaport e Dipert hanno applicato lo stesso metodo all'insegnamento della logica. Qual è il modo migliore per capire che cosa è una dimostrazione? Programmare un calcolatore in modo che possa costruirne una.
Il risultato del lavoro dei nostri tre autori è un libro semplice e autosufficiente, che introduce il lettore ai concetti di base della logica seguendo la via dell'"imparo facendo". Certo, dato il taglio introduttivo del volume, gli aspetti più ardui della logica e le applicazioni concrete all'Intelligenza Artificiale non possono esservi trattati. Tuttavia, sarebbe difficile trovare un'introduzione altrettanto chiara a concetti fondamentali e complessi come, ad esempio, quello di deduzione naturale.

A mio avviso, il volume di Shagrin, Rapaport e Dipert può essere utilizzato sia come un'introduzione alla logica in sé, sia come un primo manuale sulla dimostrazione automatica di teoremi, e come tale può essere adottato come libro di testo nei corsi di programmazione di base e avanzati, e nei corsi di Intelligenza Artificiale. Non mancano gli esercizi, che offrono al lettore un'occasione di rielaborare i concetti esposti e di valutare autonomamente il proprio grado di apprendimento.

Marco Colombetti

1
Che cos'è la logica

In senso lato, la logica è lo studio della correttezza del ragionamento. Essa produce ed esamina metodi per identificare il ragionamento corretto e quello errato, in ogni circostanza: nel nostro stesso pensiero, negli scritti altrui e nella conversazione dei nostri amici. La logica fornisce regole per determinare come si debba passare da una *credenza* (convinzione) a un'altra, ossia ci offre dei criteri per determinare quali credenze siano accettabili sulla base di altre credenze; perciò si suole anche dire che la logica è lo studio delle leggi del pensiero. Più precisamente, la logica studia le leggi del pensiero (o del ragionamento) corretto. Essa pertanto non studia il modo in cui le persone ragionano effettivamente, bensì si occupa di un modo ideale di ragionare.
Stabilire criteri di ragionamento corretto per ogni applicazione (vita quotidiana, psicologia, storia, fisica, matematica) sarebbe ovviamente un compito immane. Ogni campo di applicazione e ogni circostanza richiedono infatti criteri diversi di definizione del ragionamento corretto. In matematica esistono criteri ben precisi per determinare la correttezza di un calcolo o di una dimostrazione, ma per decidere se portare con sé l'ombrello, in base allo stato attuale del tempo e alle previsioni meteorologiche, si ricorre necessariamente a un procedimento assai meno rigidamente definito. In questo caso si potrebbe ragionare così: "Il cielo è coperto di nubi nere; allora pioverà, perciò mi conviene portare l'ombrello".
Si consideri l'enunciato seguente:

1. Alcune mele sono rosse.

A partire da questo enunciato e dalla conoscenza del fatto che (2) tutte le mele sono frutti, potremmo correttamente inferire che:

3. Alcuni frutti sono rossi.

Sarebbe invece errato concludere che:

4. Tutti i frutti sono rossi.

Come vedremo, inferire (3) da (1) e (2) è sempre un ragionamento corretto, mentre inferire (4) da (1) e (2) non lo è.

1.1 LA LOGICA DEDUTTIVA

Esiste una forma basilare di ragionamento che è accettata da tutte le scienze e da tutte le discipline, in ogni circostanza: la *logica deduttiva*. La logica deduttiva studia modalità di ragionamento che non fanno mai passare da credenze corrette a credenze errate.
Se un ragionamento è accettabile secondo i criteri della logica deduttiva, possiamo esser certi che esso è corretto in ogni circostanza. Se però un ragionamento non è accettabile in base alla logica deduttiva, non siamo autorizzati a considerarlo automaticamente scorretto e quindi a scartarlo. Infatti un tipo di ragionamento non deduttivo può essere accettabile per certe scienze o in certe circostanze.
L'argomento di questo libro è appunto la logica deduttiva, cioè l'insieme dei criteri di ragionamento accettabili da tutte le discipline in ogni circostanza. Come vedremo nei capitoli successivi, lo studio della logica deduttiva presenta molti punti di contatto con lo studio dei calcolatori e con i metodi dell'informatica. Ad esempio, elementi di logica deduttiva vengono frequentemente applicati alla programmazione dei calcolatori; inoltre i calcolatori possono essere usati con profitto per risolvere problemi di logica deduttiva.
La matematica è un esempio di disciplina che storicamente ha sempre usato solo la logica deduttiva come criterio di ragionamento corretto, senza estensioni o aggiunte. In altre parole, i criteri di ragionamento accettati in matematica sono strettamente somiglianti a quelli della logica deduttiva. Ciò non significa però che la matematica coincida con la logica deduttiva: la logica si occupa dell'identificazione dei ragionamenti corretti, mentre la matematica raramente si volge a considerare i propri strumenti di ragionamento, concentrandosi piuttosto sui loro risultati.
Buona parte dello sviluppo della logica moderna deriva da tentativi di rendere rigorosa la matematica. Ma in che cosa consiste la rigorosità? La storia dei tentativi moderni di definire la rigorosità ha inizio con i filosofi razionalisti del diciassettesimo secolo, come Cartesio e Leibniz. Essi sostennero che il ragionamento è un processo costituito da passi successivi, tutti esplicitamente espressi:

> Ci atterremo esattamente ad esso [al metodo per trovare la verità] se, per passi successivi, ridurremo le proposizioni involute ed oscure a quelle più semplici, e se poi, a partire dalla comprensione intuitiva di tutte quelle che sono assolutamente semplici, cercheremo di risalire alla conoscenza di tutte le altre mediante passaggi perfettamente analoghi. [Cartesio, *Regulae ad directionem ingenii*, regola V (1628).]

Un tale metodo era proprio quanto occorreva ai matematici che studiavano i fondamenti della propria disciplina. Poiché la logica è appunto lo studio di tali passaggi, la logica e la matematica sono diventate pressoché inseparabili (se non in pratica, almeno in teoria).

1.2 LE ARGOMENTAZIONI

Il concetto di *argomentazione* è fondamentale nella logica. L'argomentazione è una forma razionale di persuasione. È possibile persuadere una persona anche sfruttandone l'emotività, o ricorrendo all'arguzia o a bei giri di frase, oppure assumendo un'espressione amichevole. Questa forma più generale di persuasione è solitamente chiamata retorica.
La logica considera forme più limitate di persuasione, che sarebbero appropriate solo per una persona perfettamente razionale, o per un calcolatore sofisticato. Queste forme di persuasione comprendono ciò che noi chiamiamo ragioni, giustificazioni e argomentazioni, mentre escludono sorrisi, arguzie e altri strumenti che facciano appello all'emotività. In senso generale, definiamo *argomentazione* un insieme di *enunciati*, alcuni dei quali costituiscono le giustificazioni di uno degli altri.
Per esempio, un avvocato potrebbe cercare di convincere i giurati che l'imputato non può aver commesso un omicidio a Trieste la notte del 29 gennaio, perché dal 27 al 31 si trovava a Palermo. A sostegno della propria tesi, l'avvocato potrebbe citare due testimoni molto affidabili, che erano con l'imputato in quel periodo. L'argomentazione dell'avvocato potrebbe essere costituita dai seguenti enunciati:

1. Il testimone A era a Palermo con l'imputato dal 27 al 30 gennaio.
2. Il testimone B era a Palermo con l'imputato dal 28 al 31 gennaio.

Perciò

3. L'imputato era a Palermo dal 27 al 31 gennaio.

Perciò

4. L'imputato non era a Trieste la notte del 29 gennaio, quando fu commesso l'assassinio.

In questo esempio, gli enunciati (1), (2) e (3) sono forniti come motivi per credere all'enunciato (4).
In realtà la testi dell'avvocato consiste di due argomentazioni distinte: gli enunciati (1) e (2) sono forniti come giustificazioni dell'enunciato (4). In ogni argomentazione c'è un insieme di enunciati, alcuni dei quali costituiscono le giustificazioni dell'enunciato rimanente. L'enunciato rimanente è l'affermazione su cui si argomenta, ed è detto *conclusione* dell'argomentazione. Ciascuna giustificazione della conclusione è detta *premessa*.
Spesso diverse premesse rimangono sottintese. Talvolta esse sono così ovvie che non occorre esprimerle. Ad esempio, l'argomentazione dell'avvocato non esprime il fatto che Palermo e Trieste sono così distanti che l'imputato non avrebbe potuto recarsi a Trieste e tornare a Palermo senza che la sua assenza venisse notata dai testimoni. A volte, per valutare un'argomentazione, è importante non sottintendere *nessuna* premessa. In certi casi, infatti, le premesse mancanti potrebbero non essere ovvie (si supponga ad esempio che il caso venga studiato da un avvocato straniero che non conosca la geografia italiana), o addirittura potrebbero essere false (l'avvocato difensore potrebbe cercare di fuorviare i giurati). Nessuno degli esempi di questo libro avrà premesse sottintese.

INDICATORI DI PREMESSA E DI CONCLUSIONE

La prima fase della valutazione di un'argomentazione consiste nell'identificare la conclusione e nel trovare tutte le premesse. Per individuare questi elementi è spesso utile cercare determinate parole-chiave, che indicano se un enunciato è una premessa oppure una conclusione.

Indicatori di premessa. Una frase che segua una di queste espressioni è solitamente una premessa:

Poiché
Siccome
Perché
Giacché
Dato che
Posto che
Visto che
Per il fatto che
In quanto
È implicato da
È dovuto a
Deriva dal fatto che

Indicatori di conclusione. Una frase che segua una di queste espressioni è solitamente una conclusione:

Perciò
Dunque
Quindi
Allora
Di conseguenza
Per questo motivo
Ne segue che
Ciò implica che
Ciò comporta che
Ciò dimostra che
Ciò significa che
Da ciò si può dedurre che
Ne risulta che

Possiamo ora applicare ad un esempio di argomentazione questi criteri di individuazione delle premesse e della conclusione. Si consideri l'argomentazione seguente, espressa in italiano corrente:

> È noto che si può ottenere una riduzione del tasso di inflazione solo aumentando temporaneamente la disoccupazione. Dunque, ogni possibile scelta risulterebbe impopolare, giacché sia l'inflazione, sia la disoccupazione sono impopolari.

In questa argomentazione possiamo individuare un indicatore di conclusione, 'dunque', e un indicatore di premessa, 'giacché'. La conclusione di questa argomentazione risulta dunque essere:

Ogni possibile scelta risulterebbe impopolare.

Una delle premesse è:

Sia l'inflazione, sia la disoccupazione sono impopolari.

L'altra premessa, che non contiene un indicatore di premessa, è il primo enunciato; è abbastanza evidente che esso è fornito come elemento a favore della conclusione.
Il criterio degli indicatori fornisce un valido suggerimento, ma conviene verificare queste supposizioni riscrivendo l'argomentazione in una forma che evidenzi bene le premesse e la conclusione. Il risultato di questa riscrittura costituisce con ogni probabilità un'analisi corretta delle premesse e della conclusione se conserva il senso dell'argomentazione originaria. Nel nostro caso abbiamo:

1. La riduzione del tasso di inflazione aumenta la disoccupazione.
2. Sia l'inflazione, sia la disoccupazione sono impopolari.
Perciò
3. Ogni possibile scelta è impopolare.

Effettivamente, questa sembra una ricostruzione corretta del senso dell'argomentazione originaria.

VALIDITÀ E CORRETTEZZA

Sia data un'argomentazione. Come possiamo sapere se è una buona argomentazione? Dobbiamo credere alla conclusione se crediamo alle premesse? In altre parole, le premesse sono delle buone giustificazioni della conclusione?
Per verificare la bontà di un'argomentazione è importante determinare se le premesse sono vere. Delle premesse false, infatti, non costituiscono motivi validi per accettare una conclusione. Spesso tuttavia è difficile stabilire se le premesse sono vere o false. È però sempre possibile effettuare un'altra verifica, indipendentemente dal fatto che le premesse siano vere oppure false: nel ragionamento deduttivo, la conclusione deve conseguire necessariamente dalle premesse, ossia la relazione fra le premesse e la conclusione deve portare a una conclusione vera se le premesse sono vere. Prima di descrivere più in dettaglio quest'ultima verifica, conviene considerare un altro modo di esaminare le argomentazioni: anziché chiederci quando un'argomentazione è giusta, ci possiamo chiedere quando è sbagliata. Un'argomentazione può essere errata in due modi: logicamente oppure di fat-

to. Un'argomentazione è *logicamente errata* se la conclusione non consegue necessariamente dalle premesse, mentre è *errata di fatto* se una o più delle premesse è falsa.
Si potrebbe pensare che un'argomentazione possa essere errata anche in un terzo modo, cioè nel caso in cui sia falsa la conclusione. Ma ciò potrebbe avvenire solo in due modi: o l'argomentazione è logicamente errata, e allora la conclusione non consegue dalle premesse vere, oppure l'argomentazione è errata di fatto. Se l'argomentazione non è errata né logicamente né di fatto, la conclusione consegue necessariamente dalle premesse, quindi non può essere falsa.
Poiché a volte non è noto se le premesse siano vere o false, risulta utile soprattutto la verifica di *validità*, che consiste nel determinare se un'argomentazione è logicamente errata oppure no. Essa è definita nel modo seguente:

> Un'argomentazione si dice *valida* se è impossibile che la conclusione sia falsa quando le premesse sono tutte vere.

Si noti che questa definizione non afferma né implica che le premesse di un'argomentazione valida debbano essere vere, in quanto si occupa di esattezza logica, e non di esattezza di fatto; inoltre non afferma neppure che la conclusione sia vera. La definizione dice soltanto che la conclusione deve essere vera se le premesse sono vere. Chiameremo *conclusione valida* la conclusione di un'argomentazione valida.
Un'argomentazione si dice *non valida* se la conclusione può essere falsa quando le premesse sono tutte vere.
Le argomentazioni migliori sono ovviamente quelle dotate di esattezza sia logica, sia di fatto. Tali argomentazioni sono dette corrette. Definiremo dunque *corretta* un'argomentazione che è valida e ha tutte le premesse vere. Come abbiamo visto, un'argomentazione può essere *scorretta* in due modi: o perché non è valida, o perché ha almeno una premessa falsa (oppure perché si verificano contemporaneamente entrambi i casi).
Vediamo ora alcune argomentazioni che illustrano diverse possibili combinazioni di premesse e conclusioni vere e false, e classifichiamole in valide e non valide, corrette e scorrette. Per ciascun esempio riportato, si cerchi di capire bene perché è valido oppure non valido.

1. VALIDA e CORRETTA
   Tutti i cani sono mammiferi. (vero)
   Fido è un cane. (vero)

Perciò
   Fido è un mammifero. (vero)

2. NON VALIDA e SCORRETTA
   Tutte le Ford sono automobili (vero)
   La mia Fiat è un'automobile (vero)

Perciò

La mia Fiat è una Ford. (falso)

3. VALIDA ma SCORRETTA

Tutti i gatti sono cani. (falso)

Tutti i cani sono mammiferi. (vero)

Perciò

Tutti i gatti sono mammiferi. (vero)

4. NON VALIDA e SCORRETTA

Tutti i presidenti recenti degli Stati Uniti hanno abitato nella Casa Bianca. (vero)

Reagan ha abitato nella Casa Bianca. (vero)

Perciò

Reagan è un presidente recente degli Stati Uniti. (vero)

1.3 VALORI DI VERITÀ

Gli enunciati, così come le proposizioni, le affermazioni e le asserzioni, sono entità che possono essere vere o false. La verità o falsità di un enunciato è detta *valore di verità* dell'enunciato.

Determinare il valore di verità di un enunciato significa definire se l'enunciato è vero o falso. Alcuni enunciati, come

$1 = 1$

godono dell'interessante proprietà di essere sempre veri; altri enunciati, come

$1 \neq 1$

sono sempre falsi.

Esistono poi enunciati il cui valore di verità dipende, per così dire, dallo stato delle cose. Ad esempio, il valore di verità dell'enunciato

Il presidente della Repubblica ha più di 70 anni.

dipende da chi è l'attuale presidente della Repubblica.

Finora abbiamo solo detto, in modo abbastanza vago, che il valore di verità di un enunciato può essere vero oppure falso.

Conviene considerare il valore di verità come una *proprietà* di ogni enunciato: bisogna stare attenti a non confondere il valore di verità di un enunciato con l'enunciato stesso. Il valore di verità dell'enunciato 'La Terra è piatta' coincide con il valore di verità 'falso', mentre l'enunciato in sé è qualcosa di ben distinto dal valore di verità 'falso': l'essere falso è una proprietà dell'enunciato, ma non coincide con l'enunciato stesso.

Molti importanti linguaggi di programmazione dei calcolatori indicano esplicitamente i valori di verità 'vero' e 'falso' con due apposite parole-chiave, che di soli-

to sono rispettivamente 'TRUE' e 'FALSE'. D'ora in poi utilizzeremo queste due parole-chiave, in maiuscolo, per indicare i valori di verità degli enunciati. Questi due valori, TRUE e FALSE, sono spesso detti *valori booleani*, dal nome del logico inglese del diciannovesimo secolo George Boole. Le espressioni che possono assumere uno di questi valori (in particolare, gli enunciati) sono dette *variabili booleane*.

Un logico famoso, Gottlob Frege (1848-1925), giunse ad affermare che i valori di verità degli enunciati sono entità uniche ed astratte, dette il Vero e il Falso. Questo livello di astrazione, comunque, non sempre è necessario, e per svariati motivi, piuttosto tecnici che filosofici, i logici e gli informatici spesso preferiscono trattare i valori di verità TRUE e FALSE alla stregua di numeri. La notazione correntemente adottata indica TRUE con il numero 1 e FALSE con il numero 0. Questa convenzione non significa che TRUE e 1 siano la stessa cosa, né che lo siano FALSE e 0. In effetti, potremmo benissimo invertire questa corrispondenza e rappresentare FALSE con 1 e TRUE con 0. I logici tuttavia preferiscono associare 1 a TRUE e 0 a FALSE. Questa associazione ha diverse giustificazioni ma, soprattutto fra le due possibili, è la più facile da ricordare. Si potrebbe dire che un enunciato falso ha valore nullo, che niente è maggiore della verità, che la verità è maggiore della falsità, oppure che la verità è unità. Sono tutti modi di dire approssimativi, che non vale la pena di approfondire, ma che possono aiutare a ricordare correttamente la corrispondenza. In realtà, vedremo che questa associazione ha una giustificazione più profonda, legata a un'analogia fra il comportamento di 0 e 1 in aritmetica e quello di FALSE e TRUE in logica.

In ambito informatico, e in generale ove si vogliano evitare ambiguità, si preferisce usare i termini TRUE e FALSE, piuttosto che i loro corrispondenti numerici 1 e 0. In questo libro useremo sia TRUE e FALSE, sia i loro corrispondenti numerici, secondo le circostanze.

È estremamente importante distinguere chiaramente il valore di verità di un enunciato dall'enunciato stesso. Definiamo allora una notazione adatta a questo scopo. Indicheremo degli enunciati ben precisati con le lettere maiuscole, dalla 'A' alla 'O', eventualmente numerate:

A, B, C, ..., O, A1, B1, ..., O1, A2, ...

Per indicare il valore di verità di un enunciato scriveremo:

V(<enunciato>)

Ad esempio:

C = 'La Terra è piatta.'
V(C) = FALSE

'V(C)' rappresenta dunque il valore di verità dell'enunciato C. Per indicare enunciati arbitrari o non specificati, useremo le successive lettere maiuscole dell'alfa-

beto, in neretto:

P, **Q**, **R**, ..., **Z**

Vediamo alcuni esempi di uso di questa notazione:

A = 'La logica è appassionante.'
B = 'Tutti gli uccelli volano.'
V('Esistono numeri dispari.') = TRUE
V('La Terra è piatta.') = FALSE
V(B) = FALSE
Si consideri un enunciato **P** tale che V(**P**) = TRUE.
Per ogni enunciato **Q**, o V(**Q**) = TRUE o V(**Q**) = FALSE.

1.4 CONCLUSIONI

La logica è lo studio del ragionamento corretto. Una branca della logica, la *logica deduttiva*, studia un tipo di ragionamento che non conduce mai da enunciati veri a enunciati falsi.
Un'entità elementare della logica è l'*argomentazione*, un insieme di enunciati dei quali uno (la *conclusione*) è posto come conseguenza degli altri (le *premesse*). Una buona argomentazione deduttiva deve godere di due importanti proprietà. In primo luogo, tutte le sue premesse devono essere vere; in caso contrario, l'argomentazione si dice *errata di fatto*. In secondo luogo, l'argomentazione deve essere tale che la conclusione risulti vera quando si assumano vere tutte le premesse; se ciò avviene, l'argomentazione è *valida*, altrimenti è *logicamente errata*. Diremo *corretta* un'argomentazione che non sia errata né di fatto né logicamente.
Useremo le lettere maiuscole A, B, C, ..., O per indicare enunciati ben precisati, mentre useremo le lettere maiuscole in neretto **P**, **Q**, **R**, ..., **Z** per indicare enunciati arbitrari. Ogni enunciato ha un *valore di verità*, che può essere TRUE (vero) oppure FALSE (falso). A volte rappresenteremo TRUE e FALSE rispettivamente con i numeri 1 e 0. Indicheremo con V(**P**) il valore di verità di un enunciato **P**.

1.5 ESERCIZI

A. Individuare gli indicatori di premessa e di conclusione, le premesse e le conclusioni delle argomentazioni seguenti:

1. I calcolatori non possono essere intelligenti, perché non sono esseri umani.
2. I robot non possono essere persone, perché non sono in grado di pensare, mentre essere capace di pensare è una condizione necessaria per essere una persona.
3. Dio non esiste. Se esistesse, non permetterebbe la sofferenza. Ma la sofferenza esiste.

4. Potrebbero esistere altri esseri intelligenti nell'universo. Può darsi tuttavia che non ne avremo mai la certezza, in quanto essi potrebbero essere troppo lontani perché noi possiamo scoprirli.
5. Il deficit statale produce sempre inflazione, in quanto accresce l'emissione di moneta senza incrementare la produzione di beni. Orbene, l'inflazione è appunto un aumento di emissione di moneta più rapido dell'incremento di produzione di beni.
6. "Il pensiero è una funzione dell'anima immortale dell'uomo. Dio ha dato un'anima immortale ad ogni uomo e ad ogni donna, ma non ad altri animali o a macchine. Perciò nessun animale e nessuna macchina può pensare." (Turing, 1950, in Anderson, 1964.)
7. "Diversi risultati della logica matematica possono essere interpretati come limitazioni delle possibilità delle macchine a stati discreti [cioè dei calcolatori]. Il più noto di questi risultati è il teorema di Gödel, secondo il quale in ogni sistema logico sufficientemente potente è possibile formulare affermazioni che non possono essere né dimostrate né confutate entro il sistema stesso, a meno che tale sistema non sia contraddittorio. [Un essere umano è però in grado di determinare se queste affermazioni sono vere o false.]... Fin qui il teorema. Esso può essere interpretato come una prova del fatto che le macchine presentano limitazioni cui l'intelletto umano non è soggetto." (Turing, 1950, in Anderson, 1964.)
8. "Certamente il sistema nervoso non è una macchina a stati discreti. Un piccolo errore nell'informazione relativa all'ampiezza di un impulso nervoso in ingresso a un neurone può alterare notevolmente l'ampiezza dell'impulso di uscita. Se ne può dedurre che, stando così le cose, appare impossibile la simulazione del comportamento del sistema nervoso mediante una macchina a stati discreti." (Turing, 1950, in Anderson, 1964.)
9. "Il dibattito filosofico corrente verte sul fatto che la mente e gli stati mentali sono così nettamente diversi dal corpo e dagli stati fisici, che una connessione causale fra essi appare inconcepibile. È indubbio che, se la mente e gli eventi mentali sono esclusivamente ciò che appaiono essere all'introspezione, e se il corpo e gli eventi fisici sono proprio come l'intelligenza e il buon senso ci portano a concepirli, allora tale netta differenza esiste. Ciò dovrebbe farci concludere che, per quanto certi eventi mentali possano essere strettamente correlati a certi eventi fisici, non può sussistere fra essi un legame di causalità." (Broad, 1962.)

B. Scrivere due semplici argomentazioni:
   1. La prima non valida ma con conclusione vera.
   2. La seconda valida ma con conclusione falsa.

C. Completare la tabella a pag. 23 con i simboli

T : TRUE	NV : non valida
F : FALSE	C : corretta
V : valida	S : scorretta
? : informazioni insufficienti	

Ad esempio:

I T:premesse tutte vere F:almeno una premessa falsa	II Argomentazione valida o non valida	III Argomentazione corretta o scorretta	IV Conclusione vera o falsa
1. T	V	C	□
2. F	V	□	□

SOLUZIONE

1. La conclusione di un'argomentazione valida avente premesse vere (cioè di un'argomentazione corretta) deve essere vera; perciò bisogna mettere T nella colonna IV.
2. Un'argomentazione valida avente una premessa falsa è scorretta; perciò bisogna mettere S nella colonna III. Una tale argomentazione può avere conclusione vera oppure falsa; perciò bisogna mettere '?' nella colonna IV.

	Premesse	**Argomentazione**		**Conclusione**
3.	F	□	S	□
4.	F	NV	□	□
5.	T	NV	□	□
6.	T	□	S	□
7.	□	V	S	F
8.	□	V	□	F
9.	□	V	□	T
10.	□	□	C	T
11.	□	NV	□	T
12.	□	□	S	T
13.	□	NV	□	F
14.	□	□	S	F
15.	F	□	□	F
16.	F	□	□	T
17.	T	□	□	T
18.	T	V	□	□
19.	□	□	C	□
20.	T	□	□	□
21.	□	V	□	□
22.	□	□	S	□
23.	□	□	□	F
24.	T	□	□	F

D. Abbiamo visto che alcune parti di un ragionamento possono contenere più di un'argomentazione. Nei due gruppi di enunciati che seguono l'esempio, si contino le argomentazioni distinte e si usino i numeri degli enunciati per indicare le premesse e le conclusioni delle diverse argomentazioni. Esempio:

1. I calcolatori non sono fatti di materia vivente.
2. Solo la materia vivente è dotata di sensibilità.
3. Dunque i calcolatori sono privi di sensibilità.
4. Avere sensibilità è però necessario per pensare in senso lato.
5. Perciò i calcolatori non sono neppure in grado di pensare in senso lato.

Soluzione:
Numero di argomentazioni: 2
Struttura della prima argomentazione:
Premesse: 1, 2
Conclusione: 3
Struttura della seconda argomentazione:
Premesse: 3, 4
Conclusione: 5

Gruppo A
1. La geometria è una branca della matematica.
2. Perciò la matematica abbraccia più argomenti della geometria.
3. La matematica è una branca della logica.
4. Dunque la logica abbraccia ancor più argomenti della matematica.
5. A maggior ragione la logica comprende molti più argomenti di quanti ne comprenda la geometria.

Gruppo B
1. Tutti i numeri primi sono dispari.
2. Dunque nessun numero pari è primo.
3. Alcuni numeri dispari sono divisibili per 3.
4. Dunque non tutti i numeri dispari sono primi.

2
Logica, calcolatori e algoritmi

Storicamente, l'evoluzione dei calcolatori è strettamente legata alla logica. Attualmente si è portati a considerare la storia dei calcolatori solo in termini di aumento di velocità e di efficienza nel compiere calcoli matematici. Esaminando il passato, tendiamo a vedere solo una successione di strumenti di calcolo, quali l'abaco, il regolo, la macchina addizionatrice, la calcolatrice tascabile e il calcolatore, che ci sembra siano stati tutti progettati essenzialmente per effettuare calcoli matematici. L'interpretazione dei calcolatori come strumenti di calcolo matematico è però storicamente incompleta. Alcuni dei primi progetti di calcolatori non miravano all'effettuazione di calcoli matematici, bensì alla soluzione del problema logico di definire il ragionamento deduttivo corretto. Alcuni dei primi calcolatori furono dunque progettati per generare le conclusioni logicamente valide di determinate premesse, oppure per verificare la validità di un ragionamento.

2.1 RAGIONAMENTO AUTOMATICO

L'uso di una macchina per determinare la correttezza di un ragionamento fu un progresso del tutto naturale. In primo luogo, lo sviluppo di metodi simbolici per rappresentare problemi logici permise di sottoporre tali problemi a una macchina: senza una formalizzazione chiara e sintetica del ragionamento, sotto forma di *logica simbolica*, sarebbe stato impossibile comunicare tali problemi a delle macchine. Analogamente, sarebbe stato impossibile sviluppare la matematica e proporre problemi matematici ai calcolatori, senza usare rappresentazioni simboliche concise e sistematiche dei problemi matematici, come ad esempio le cifre arabe e i simboli '+' e '−'.
In secondo luogo, la natura del ragionamento corretto è un problema antico e di vitale importanza in filosofia e in altre discipline.
Con la ripresa degli studi su questo argomento, nel diciassettesimo secolo, da parte di Cartesio, Leibniz e altri, e con le ipotesi speculative sulla natura del ragiona-

mento corretto, si pose il problema della costruibilità di macchine in grado di ragionare, o almeno di verificare la validità di ragionamenti formulati dall'uomo. Delle macchine che fossero in grado di valutare la correttezza di un ragionamento umano, pur non essendo esse stesse capaci di ragionamento creativo, sarebbero estremamente utili in matematica e in altri campi, in quanto potrebbero verificare la validità di ragionamenti lunghi o complicati. A causa di questi due fattori, la storia dei calcolatori è ricca di tentativi di costruire macchine in grado di ragionare o di verificare ragionamenti.
Uno dei tentativi più fantasiosi e sorprendenti di automatizzare il ragionamento si trova nell'opera del mistico e prete spagnolo del tredicesimo secolo Raimondo Lullo (Ramón Lull). Non vi fu nulla di monotono nella vita di questo bizzarro personaggio, ad eccezione forse dei suoi numerosi e voluminosi libri. (Per ulteriori informazioni sulla vita e l'opera di Lullo si consiglia di leggere il primo capitolo di *Logic Machines and Diagrams* di Martin Gardner.) Lullo è considerato uno dei primi autori che abbiano cercato di definire una notazione per la soluzione di problemi logici, nonché forse il primo che abbia tentato di costruire uno strumento meccanico in grado di risolverli. Le macchine che costruì a questo scopo erano costituite da ruote concentriche a settori; facendo ruotare una di esse, si ottenevano diverse combinazioni di simboli. Tali ruote erano molto simili a quelle usate al giorno d'oggi per determinare il momento più adatto per certi tipi di semina, in base al terreno, alla posizione, o ad altri parametri.
Il filosofo, matematico e diplomatico tedesco Gottfried Wilhelm Leibniz (1646-1716) fu molto colpito dalle idee di Lullo. Leibniz concepì progetti grandiosi per migliorare le modalità del ragionamento; propose, ad esempio, l'uso di un linguaggio logico universale. Questo linguaggio doveva essere progettato in modo che ogni enunciato mostrasse chiaramente il proprio contenuto logico, analogamente alle equazioni matematiche, che sono universalmente comprensibili. Esso doveva inoltre evidenziare in modo chiaro e non ambiguo le relazioni logiche fra gli enunciati.
Parallelamente al progetto di un linguaggio logico universale, Leibniz proponeva la ricerca dei metodi che un ragionamento deve seguire per risultare corretto. L'insieme di questi metodi doveva costituire ciò che Leibniz chiamò *calculus ratiocinator*. Ricorrendo ad esso, qualunque persona avrebbe potuto, ragionando, pervenire alle medesime conclusioni in maniera praticamente automatica, allo stesso modo in cui due contabili finiscono necessariamente col convenire sul risultato di un'addizione. Leibniz suggerì addirittura che si sarebbero potute usare delle macchine per trattare alcune parti gravose di concatenazioni lunghe o complicate di ragionamenti. Leibniz non costruì una tale macchina logica, ma riuscì effettivamente a costruire, nel 1673, una calcolatrice meccanica per effettuare calcoli aritmetici. Leibniz stesso non riuscì a sviluppare significativamente i progetti paralleli di un linguaggio logico universale e di un calcolo del ragionamento; tuttavia questi progetti ambiziosi costituirono per secoli uno stimolo per i filosofi e i matematici e, più recentemente, per gli informatici.
La prima macchina logica funzionante può forse essere attribuita allo scrittore, politico e inventore inglese del diciottesimo secolo Charles Stanhope (1753-1816).

Stanhope inventò un congegno meccanico che chiamò "Dimostratore". Come la maggior parte dei primi calcolatori logici, esso veniva fatto funzionare muovendo manopole o levette per definire vere certe premesse; degli indicatori mostravano poi tutte le conclusioni che seguivano logicamente dalle premesse. Era inoltre possibile usare il Dimostratore anche per individuare le conclusioni rese probabili dalle premesse.
Una macchina logica molto più complessa fu inventata dall'economista e logico inglese William Stanley Jevons (1835-1882). Il calcolatore di Jevons divenne famoso come "pianoforte logico", a causa della sua tastiera d'ingresso. Jevons, che visse quasi un secolo dopo Stanhope, ebbe il vantaggio di poter usare un raffinato sistema di notazione logica inventato da George Boole intorno agli anni 1830 e 1840, ora noto, in forma modificata, come *algebra di Boole*. La macchina logica di Jevons, pur rappresentando un notevole progresso rispetto al Dimostratore di Stanhope, aveva diverse importanti limitazioni: poteva trattare solo con grande difficoltà gli enunciati cosiddetti "particolari", della forma 'Alcuni A sono B' (come 'Alcuni sempreverdi sono pini'), e non era in grado di trattare argomentazioni riferentisi a più di quattro classi distinte.
Nella nostra discussione abbiamo finora separato il concetto di macchina logica, cioè in grado di risolvere problemi logici, da quello di macchina calcolatrice aritmetica, in grado di effettuare addizioni, moltiplicazioni, eccetera. Non abbiamo perciò considerato alcune delle macchine, spesso assai sofisticate, che furono progettate esclusivamente per calcoli aritmetici, come la macchina addizionatrice inventata dal filosofo francese Blaise Pascal. Una macchina, tuttavia, è degna di menzione, pur essendo stata progettata in origine per risolvere problemi aritmetici: la "macchina analitica" di Charles Babbage, un inglese che sviluppò questo progetto intorno agli anni 1830 e 1840. Babbage non riuscì a completare la macchina analitica nel corso della propria vita, ma un parziale completamento fu operato da suo figlio dopo la sua morte. La caratteristica più notevole della macchina di Babbage consiste nel fatto che la sua impostazione globale presentava alcune analogie con i moderni calcolatori. Come questi ultimi, infatti, la macchina analitica doveva essere un calcolatore multiscopo, in grado di effettuare qualunque operazione le venisse indicata; era programmabile mediante schede, aveva una memoria ed era in grado di decidere quale fosse la successiva operazione da effettuare.
Le idee di Babbage non furono molto apprezzate dal mondo accademico inglese, ma egli trovò un'assidua sostenitrice in Ada Lovelace, la bella figlia (illegittima) del poeta romantico inglese Byron. Un recente linguaggio di programmazione, Ada, è stato così chiamato in onore di questa famosa sostenitrice delle possibilità e del futuro dei calcolatori.
Verso la fine del diciannovesimo secolo furono apportati miglioramenti ai calcolatori logici meccanici come quello di Jevons e furono progettate le prime macchine logiche elettriche. Risale ad allora anche il primo contributo statunitense, con le macchine e i progetti del filosofo Charles S. Peirce (1839-1914) e del suo allievo Allan Marquand (1853-1924). Ma lo sviluppo di calcolatori sofisticati per risolvere problemi logici generali dovette attendere la rapida evoluzione dei circuiti elet-

tronici di commutazione, nel secondo dopoguerra. Le circostanze di questo sviluppo sono note: l'evoluzione, sorprendentemente rapida, da interruttori meccanici a elettrici (come i relè), a elettronici (come i tubi a vuoto), fino ai circuiti elettronici integrati (transistor e chip).
Benché tutti questi recenti sviluppi siano indubbiamente dovuti alla ricerca di una sempre maggior velocità di calcolo aritmetico e di elaborazione di dati, piuttosto che al perseguimento dello scopo essenzialmente logico di identificare il ragionamento corretto, permane un vivo interesse per le applicazioni dei calcolatori alla logica. Ben presto, infatti, è risultato chiaro che diversi aspetti del progetto di calcolatori elettronici sono perfettamente analoghi a certi tipi di analisi propri della logica. Il funzionamento dei calcolatori moderni si basa sul controllo del flusso di segnali elettrici entro i loro circuiti. Determinare se un segnale elettrico sia presente o assente è analogo a determinare se un enunciato valga TRUE o FALSE: in entrambi i casi sono possibili due valori. La stretta relazione fra il controllo di tipo presente/assente (o 'alto/basso') sui segnali elettrici e le operazioni logiche di tipo TRUE/FALSE è studiata dalla *teoria della commutazione*, sviluppata da Claude Shannon negli anni 1937 e 1938. Questo argomento viene approfondito nell'Appendice A.
A partire dagli anni Cinquanta, gli informatici si sono sempre più occupati del problema, in origine filosofico e logico, di determinare se un'argomentazione è deduttivamente valida e, in caso affermativo, di fornirne la dimostrazione. In effetti, uno dei primi programmi nel campo dell'Intelligenza Artificiale fu il 'Logic Theorist', che era capace di dimostrare teoremi in logica enunciativa. (Per ulteriori informazioni su questo e su altri programmi recenti, si vedano Slagle, 1971, e Rich, 1986.)
Qualunque calcolatore moderno, dal grande calcolatore (o "mainframe") al microcalcolatore, può essere programmato, nella maggior parte dei casi, per determinare se un'argomentazione è valida, e anche per mostrare *perché* la conclusione segue validamente dalle premesse. Inoltre, un calcolatore può essere programmato per verificare se una dimostrazione da noi prodotta è conforme alle regole di inferenza definite. Risulta però che, per ragioni estremamente interessanti (discusse nel Capitolo 14 e nell'Appendice B), i calcolatori non possono in generale decidere se una data conclusione segue validamente da determinate premesse oppure no. Alla fine della maggior parte dei capitoli di questo libro forniremo consigli pratici per trasformare in effettivi programmi per calcolatore quanto avremo esposto.

2.2 CALCOLATORI E RAGIONAMENTO

I calcolatori attuali pensano? È possibile che un calcolatore pensi? Sono domande ricorrenti e difficili, cui non possiamo dare risposte esaurienti in questa sede. Questi problemi sono trattati da un settore dell'informatica, detto Intelligenza Artificiale, e da un settore della filosofia ad essa parzialmente sovrapposto, la filosofia della mente.

La capacità di pensare di un calcolatore è collegata, e secondo alcuni è addirittura coincidente, con la sua capacità di imitare qualche aspetto del comportamento comunemente considerato frutto dell'attività del pensare. Ad esempio, potremmo essere tentati di affermare che un calcolatore pensa, se esso sapesse rispondere in modo appropriato a delle domande correnti ("Che ore sono?", "Come va oggi?"), oppure se fosse in grado di sostenere una conversazione.
Un altro tipo di comportamento che potrebbe portarci ad affermare che un calcolatore è in grado di pensare sarebbe qualche dimostrazione di capacità di ragionare. Potremmo considerare una prova della facoltà di ragionare, la capacità di:

1. Determinare correttamente se un'argomentazione è valida o no.
2. Dedurre da un enunciato altri enunciati che seguano validamente da esso.
3. Se un'argomentazione è valida, produrre una derivazione che ne mostri la validità.

In questo libro studieremo come si possa istruire un calcolatore per fargli svolgere queste attività; esamineremo perciò alcuni ambiti in cui si potrebbe affermare che i calcolatori sono in grado di ragionare.
Il problema di definire se i calcolatori siano o meno capaci di pensare o di provare emozioni è difficile e controverso. Tuttavia, il ragionamento è una parte di ciò che noi chiamiamo pensiero. Ebbene, il ragionamento potrebbe essere proprio quel tipo di attività che i calcolatori sono in grado di svolgere, come vedremo nei capitoli seguenti.
Per una trattazione più approfondita di questi affascinanti problemi, si consiglia di consultare uno dei seguenti libri sull'Intelligenza Artificiale:

Anderson, A.R. (ed.), *Minds and Machines*, Englewood Cliffs, N.J., Prentice-Hall, 1964
Boden, M., *Artificial Intelligence and Natural Man*, New York, Basic Books, 1977
Colombetti, M., *Le idee dell'Intelligenza Artificiale*, Milano, Mondadori, 1985
Dennett, D.C., *Brainstorms: Philosophical Essays on Mind and Psychology*, Montgomery, Vt., Bradford Books, 1978
Dreyfus, H., *What Computers Can't Do: The Limits of Artificial Intelligence*, New York, Harper and Row, 1979
Haugeland, J. (ed.), *Mind Design: Philosophy, Psychology, Artificial Intelligence*, Cambridge, Mass., M.I.T. Press, 1981
Hofstadter, D.R., *Gödel, Escher, Bach: An Eternal Golden Braid*, New York, Basic Books, 1979
McCorduck, P., *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*, San Francisco, Freeman, 1979
Raphael, B., *The Thinking Computer: Mind Inside Matter*, San Francisco, Freeman, 1976
Rich, E., *Intelligenza Artificiale*, Milano, McGraw-Hill, 1986
Weizenbaum, J., *Computer Power and Human Reason: From Judgment to Calculation*, San Francisco, Freeman, 1976
Winston, P.H., *Artificial Intelligence*, Reading, Mass., Addison-Wesley, 1977

2.3 COME FUNZIONA UN CALCOLATORE

I moderni calcolatori elettronici digitali (vedi Figura 2.1) sono concepiti per effettuare calcoli ed elaborare informazioni a velocità elevatissima. I calcoli che un calcolatore è in grado di effettuare vanno dalla semplice somma di due numeri a una sequenza di centinaia di operazioni come moltiplicazioni, radici quadrate e funzioni trigonometriche. Come possiamo vedere anche dagli estratti conto o dalle lettere "personalizzate" che riceviamo per posta, i calcolatori sono anche capaci di memorizzare, elaborare e selezionare informazioni contenute in milioni di registrazioni. Questo secondo insieme di compiti svolti dai calcolatori è spesso chiamato elaborazione di dati, o di informazioni.

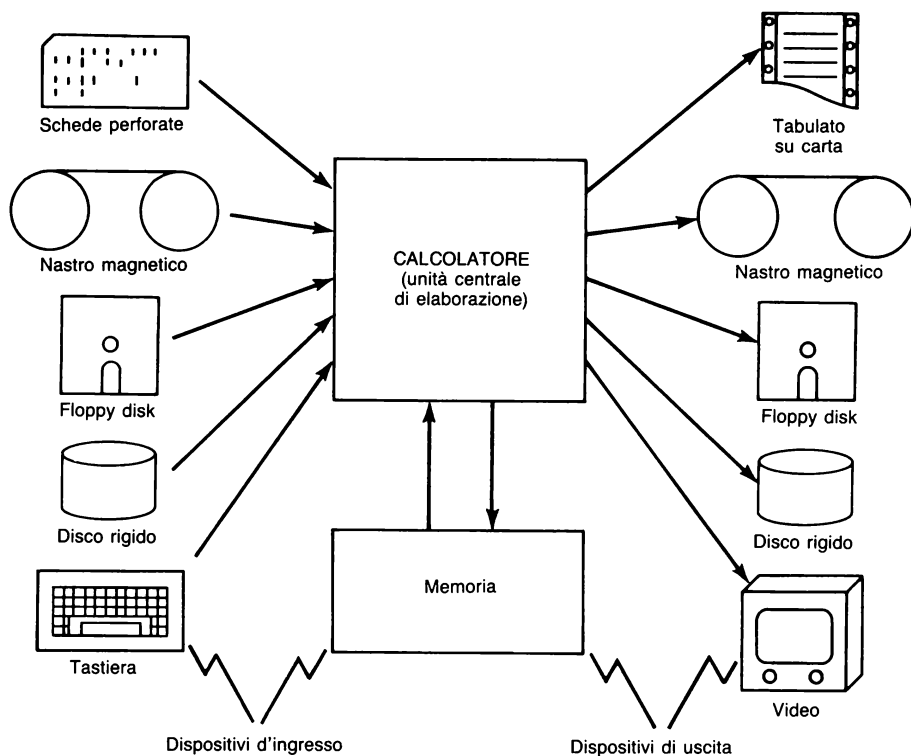



Figura 2.1 Tipica struttura di un sistema

I LINGUAGGI DI PROGRAMMAZIONE

È ormai fin troppo noto che i calcolatori sono privi di immaginazione: senza istruzioni opportune, non sono in grado di effettuare i calcoli o le operazioni che vorremmo far loro compiere. Perciò il compito fondamentale delle persone che lavorano con i calcolatori consiste nel fornire queste istruzioni.
Una volta che siamo elettronicamente collegati con il calcolatore, come ci dobbiamo esprimere? Questa fase della comunicazione con un calcolatore è appassionante, ma complessa. Se vogliamo che il calcolatore effettui per noi qualche operazione, dobbiamo descrivergli in ogni dettaglio ciò che deve fare. Nella stragrande maggioranza dei casi, un calcolatore non capirebbe quasi niente di un dialogo in italiano corrente.
Se battessimo alla tastiera "somma i numeri 5 e 8", quasi sicuramente il calcolatore non saprebbe che cosa fare, a meno che non fosse stato preparato proprio per questo tipo di frase. Probabilmente emetterebbe una segnalazione di errore, oppure non farebbe assolutamente niente. Ne ricaviamo che, almeno al giorno d'oggi, i calcolatori non sono in grado di capire qualsiasi frase in una lingua umana, anche se tale frase è perfettamente comprensibile per qualunque persona che conosca tale lingua. Al contrario, i calcolatori sono progettati per comprendere solo alcune porzioni limitate, e di aspetto talvolta innaturale, della lingua inglese, dette linguaggi di programmazione. Spesso tali linguaggi presentano una certa somiglianza con l'inglese, ma utilizzano solo una piccola parte delle parole e delle costruzioni appartenenti a tale lingua.
È spesso sorprendente ciò che un calcolatore fa quando ha ricevuto istruzioni in un linguaggio di programmazione: esegue né più né meno quanto specificato dalle istruzioni. Un calcolatore non smette di fare ciò che gli è stato ordinato, neanche se si tratta di qualcosa che non ha termine, o è privo di scopo, o non assomiglia a ciò che si voleva che facesse. Al contrario, come ben sappiamo, gli esseri umani a volte possono ignorare, correggere o cambiare le istruzioni che ricevono.

I LINGUAGGI MACCHINA

Fra i linguaggi per comunicare con i calcolatori, i più oscuri sono i cosiddetti *linguaggi macchina*. Essi permettono soltanto di far eseguire al calcolatore le singole operazioni elettroniche necessarie al conseguimento del risultato voluto. Il fastidioso compito del programmatore consiste allora nello specificare in ogni minimo dettaglio ciò che il calcolatore deve fare per effettuare un'operazione. Consideriamo come esempio la somma di due numeri in base dieci. Se stessimo comunicando con il calcolatore in linguaggio macchina, non potremmo scrivere un'istruzione del tipo "somma i due numeri interi 18 e 37". Il calcolatore non riconoscerebbe il significato dell'operazione di somma, non saprebbe che cosa sono i numeri interi e non saprebbe come utilizzare i simboli '18' e '37'.
Per scrivere in linguaggio macchina il programma (cioè l'insieme di istruzioni)

adatto per il calcolatore, dovremmo descrivere come si sommano due numeri interi. Sembra un compito semplice. Bisogna però ricordare che il calcolatore non conosce affatto l'operazione di addizione. Per farci un'idea delle istruzioni che dovremmo fornire al calcolatore, consideriamo il modo in cui effettuiamo manualmente la somma. Dapprima incolonniamo i due addendi così:

18
37

Poi cominciamo da destra ... Improvvisamente, la descrizione di questa operazione apparentemente semplice diventa davvero complicata.

I LINGUAGGI DI PROGRAMMAZIONE DI ALTO LIVELLO

La descrizione della somma di due numeri in linguaggio macchina è estremamente lunga. Possiamo cominciare a capire che se, comunicando fra noi o con i calcolatori, dovessimo ogni volta definire con tutti questi dettagli le operazioni da effettuare, ci complicheremmo la vita e ne sprecheremmo gran parte. Gli esseri umani hanno risolto da tempo questo problema, inventando una parola che descrive in forma abbreviata questo procedimento: la parola "somma". Ad essa è collegato il comando "somma ...!", che ci fa eseguire il procedimento che abbiamo cominciato ad esaminare un attimo fa. Anziché:

Incolonna *n* e *m* e allineane le ultime cifre...

diciamo:

Somma *n* e *m*.

Tutti sappiamo che cosa significa, e molti di noi sono ragionevolmente abili nell'eseguire il procedimento indicato in questa utilissima forma abbreviata.
Gli informatici hanno inventato modalità di comunicazione con i calcolatori che evitano l'eccesso di dettagli imposto dall'uso del linguaggio macchina: si tratta dei *linguaggi di alto livello*.
Tali linguaggi ricorrono al trucco già da tempo inventato dall'uomo: certe parole o simboli sono usate come abbreviazioni per indicare procedimenti piuttosto complicati, ma di uso frequente. Quasi tutti i linguaggi di programmazione di alto livello possiedono abbreviazioni per le operazioni aritmetiche più comuni, come la somma, la moltiplicazione e la divisione, nonché per altre operazioni.
Attualmente esistono molti linguaggi di programmazione (termine con il quale indicheremo d'ora in poi solo i linguaggi di alto livello). Tutti rappresentano certi procedimenti di uso frequente in forma abbreviata, o mediante simboli (*, /, +, $, %, ecc.), o mediante parole (dette *parole-chiave*), simili alle corrispondenti pa-

role inglesi (DIV, IF, NOT, ecc., per "divisione", "se", "non", ecc.). Questi linguaggi sono molto diversi fra loro, perché ciascuno è concepito per facilitare la stesura di certi tipi di programmi.

2.4 ALGORITMI

Il fondamento della comunicazione con i calcolatori, qualunque sia il linguaggio prescelto, è il concetto di algoritmo: una descrizione dettagliata, passo per passo, di un procedimento che un calcolatore o una persona seguirebbe per risolvere un problema o per svolgere un compito. Per capire meglio che cos'è un algoritmo, si consideri la seguente istruzione, apparentemente semplice:

Cuocete i piselli surgelati.

Questa istruzione generale può essere dettagliata in una sequenza di istruzioni più specifiche: fate bollire dell'acqua in un tegame, versatevi i piselli, coprite il tegame e lasciate sobbollire per qualche minuto. Già, ma quanta acqua? Quanti piselli? Come si deve regolare il fornello per lasciar "sobbollire"? Per quanti minuti? Cuochi diversi daranno risposte diverse a queste domande.
Analogamente, musicisti diversi, seguendo lo stesso spartito, eseguiranno lo stesso brano musicale in modi diversi, dando spesso luogo ad esiti assai differenti. Il cuoco o il musicista devono sopperire alla mancanza di precisione e di dettagli insita nella ricetta o nello spartito, cioè nelle istruzioni. Una sequenza di istruzioni che fosse esauriente in ogni dettaglio non permetterebbe all'esecutore di scostarsi dalla procedura indicata. Una tale sequenza di istruzioni per l'esecuzione di un compito è una *procedura automatica.*

PROCEDURE AUTOMATICHE

Un *algoritmo* è un tipo particolare di procedura automatica: è una sequenza, finita e dettagliata passo per passo, di istruzioni per l'esecuzione di un compito, la quale gode delle due proprietà seguenti: (1) una volta fornita l'informazione richiesta, il compito deve poter essere portato a termine in un numero finito di passi e con una quantità finita di risorse; (2) la sequenza di istruzioni è meccanica e priva di ambiguità, nel senso che la sua esecuzione non deve richiedere né creatività né informazioni aggiuntive.
Problemi apparentemente molto complessi, o che sembrano richiedere una certa discrezionalità interpretativa, possono a volte essere espressi mediante algoritmi sorprendentemente semplici. Viceversa, dei compiti semplicissimi, che svolgiamo quotidianamente senza pensarci troppo, possono essere estremamente difficili da esprimere in forma algoritmica. Come abbiamo visto, l'istruzione "somma n e

m'' non è sufficientemente esplicita se la persona o la macchina che la deve eseguire non conosce il concetto di somma. Per produrre un algoritmo di somma più esplicito dobbiamo fornire istruzioni più precise e dettagliate. Un tale algoritmo sarebbe costituito da una sequenza dettagliata di istruzioni, che qualcuno privo di creatività e del concetto di somma potrebbe eseguire ottenendo la somma corretta. Per definire le istruzioni da fornire al calcolatore, dovremo spesso considerare dapprima il modo in cui noi stessi effettueremmo tale operazione. Dovremo insomma esplicitare il procedimento di somma di due numeri, ormai diventato per noi del tutto naturale.
Un metodo molto utile per creare un algoritmo è l'approccio ''*discendente*'' (dal generale al particolare), al quale ricorreremo spesso. Impiegando questo metodo, dapprima si determinano, a grandi linee e nell'ordine opportuno, i passi che appaiono necessari all'esecuzione del compito. Si ritorna poi su ciascuno di questi passi, rendendolo sufficientemente preciso e dettagliato affinché l'esecutore (umano o meccanico) delle istruzioni sappia esattamente cosa fare ad ogni passo. Questa fase del progetto di un algoritmo è detta *raffinamento per passi successivi*.
Come primo passo di un'analisi *discendente*, potremmo raffinare l'istruzione ''somma n e m'' in quattro istruzioni:

1. Scrivi n.
2. Scrivi m.
3. Calcolane la somma.
4. Scrivi il risultato di tale somma.

Una tale rappresentazione a grandi linee delle azioni da compiere è detta *programma principale*. Spesso i passi del programma principale devono essere resi più espliciti. L'espansione dettagliata di un passo del programma principale è un *sottoprogramma*.
L'istruzione 1 è sufficientemente precisa, mentre la 2 può essere ulteriormente raffinata:

2.1. Scrivi le unità di m sotto le unità di n.
2.2. Scrivi le decine di m sotto le decine di n.

e così via.
Ovviamente, l'istruzione fondamentale è la 3, che può essere raffinata nel modo seguente:

3.1. Cerca la somma delle unità in una tabella (che contiene regole del tipo $0 + 0 = 0$, $0 + 1 = 1$, ..., $2 + 3 = 5$, ..., $9 + 9 = 18$).
3.2. Scrivi questa somma delle unità sotto le unità di n e di m.
3.3. Se questa somma delle unità è maggiore di 9, scrivi '1' in cima alla colonna delle decine.
3.4. Trova la somma delle cifre della prossima colonna a sinistra (colonna delle decine).

e così via. Anche l'istruzione 3.4 deve essere raffinata, giacché la tabella usata in 3.1 contiene presumibilmente soltanto somme di due numeri, mentre nella colonna delle decine ci sono tre cifre, se il passo 3.3 viene eseguito.
Il passo 3.3 è un esempio di decisione che deve essere presa durante l'esecuzione dell'algoritmo; non è però una decisione che richieda creatività o informazioni aggiuntive, perché è completamente specificata.

2.5 DIAGRAMMI DI FLUSSO

L'algoritmo precedente è stato scritto in italiano. Un programma è un modo di descrivere un algoritmo per un esecutore. Se l'esecutore è umano, conviene fornire le istruzioni in italiano, o comunque in un linguaggio naturale. Per un calcolatore occorre invece usare un programma scritto in un linguaggio di programmazione, come il BASIC o il Pascal.
Prima di scrivere effettivamente un programma, si può rappresentare l'algoritmo in forma grafica, mediante un *diagramma di flusso*. Un diagramma di flusso è un grafo costituito da punti ("nodi") collegati da frecce. I nodi rappresentano i passi dell'algoritmo, mentre le frecce rappresentano l'ordine in cui i passi devono essere eseguiti, ossia rappresentano il cosiddetto *flusso di controllo* (onde la denominazione "diagrammi di flusso").
In un diagramma di flusso sono particolarmente importanti i nodi di *inizio* e di *terminazione* (se è previsto che il programma si concluda). Solitamente, tali nodi vengono rappresentati, in un diagramma di flusso, mediante le parole-chiave 'START' (inizio) e 'STOP' (terminazione), racchiuse entro cornici arrotondate:

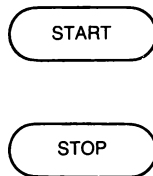



Figura 2.2 Nodi di inizio e terminazione

Di fondamentale importanza in un diagramma di flusso sono i *nodi esecutivi*, di forma rettangolare, contenenti istruzioni che rappresentano operazioni da effettuare (ad esempio, una somma). La Figura 2.3 illustra degli esempi di nodi esecutivi.

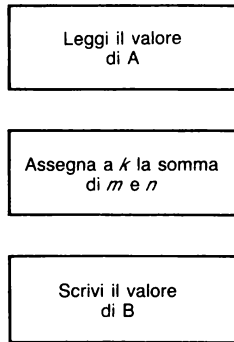



Figura 2.3 Nodi esecutivi

In alcune modalità di rappresentazione dei diagrammi di flusso si usano cornici di forma diversa per distinguere determinati tipi di operazioni, e in particolare le operazioni di *ingresso* (lettura) e di *uscita* (scrittura). Per semplicità, tuttavia, noi useremo la stessa cornice rettangolare per tutti i tipi di nodi esecutivi.
Esistono infine i nodi di *decisione* (*test*), i quali contengono enunciati che possono essere veri o falsi, oppure, a volte, domande la cui risposta può essere "sì" o "no". Questi test sono racchiusi entro un rombo e hanno due frecce di uscita: il ramo TRUE (oppure "sì") e il ramo FALSE (oppure "no"). La Figura 2.4 illustra degli esempi di nodi di test.

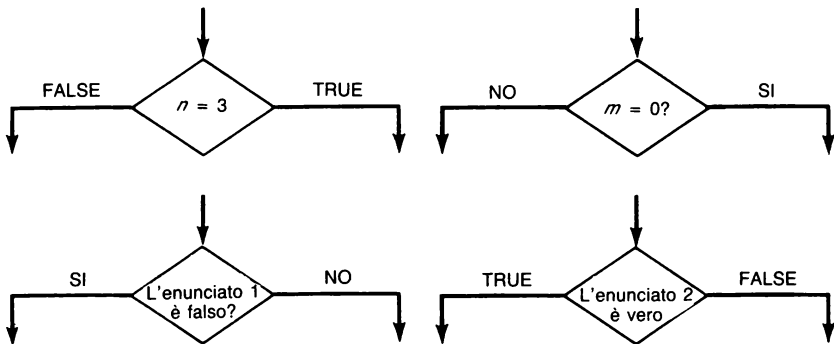



Figura 2.4 Nodi di test

Poiché i nodi di test hanno due frecce di uscita, si dice che in corrispondenza di essi il flusso di controllo *si ramifica*. Tali ramificazioni sono importanti, non già perché effettuino calcoli o compiano azioni rilevanti, bensì perché determinano

qual è, fra due passi in alternativa, il prossimo passo da eseguire.
Un diagramma di flusso è costituito da diversi rettangoli esecutivi e rombi di test, compresi fra un nodo di START e uno o più nodi di STOP. Tutti i nodi sono collegati da linee, la cui direzione è indicata da una freccia. Il diagramma di flusso più semplice possibile è allora quello che non fa assolutamente niente, salvo iniziare e terminare:

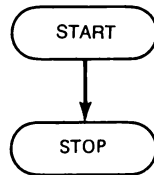



Figura 2.5 Il diagramma di flusso più semplice possibile

Si consideri questo diagramma di flusso, un po' più significativo del precedente:

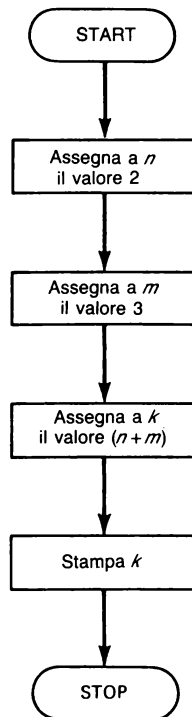



Figura 2.6 Diagramma di flusso per sommare 2 e 3

Questo programma somma 2 e 3 e stampa '5'. Si noti che in questo diagramma di flusso si è assunto di avere a disposizione una procedura per sommare due numeri; il diagramma di flusso, infatti, non descrive come si esegua la somma. Solitamente nei diagrammi di flusso, così come in italiano e nei linguaggi di programmazione, si assume che tali semplici procedure aritmetiche siano già disponibili.
Due rettangoli esecutivi molto frequenti sono quelli contenenti le parole-chiave 'INPUT' e 'OUTPUT'. INPUT significa "ricevi in ingresso" e serve per ricevere un'informazione o dall'utente, o da un luogo in cui l'informazione è memorizzata. OUTPUT significa "fornisce in uscita" ed effettua la comunicazione di un'informazione.
Nell'esempio appena visto, al termine il programma comunica un valore, che è il risultato della somma di 2 e 3. Possiamo modificare tale programma, in modo che riceva due numeri qualsiasi e ne fornisca la somma:

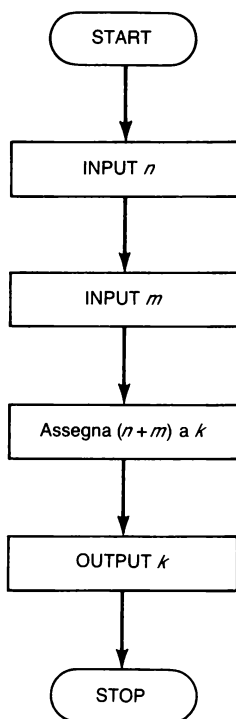



Figura 2.7 Diagramma di flusso per sommare due numeri qualsiasi

Ora il programma chiederà un'informazione all'utente ogni volta che raggiungerà un'istruzione di INPUT. Questo programma lo farà due volte. Se programmiamo un calcolatore per eseguire questo diagramma di flusso, il calcolatore si fermerà alla prima istruzione di INPUT e aspetterà un'informazione riguardo al valore da assegnare alla variabile *n*. Quando avrà ricevuto un valore, passerà al terzo passo, aspettando di conoscere il valore di *m*. Poi procederà autonomamente, calcolando il valore di *k* (somma di *m* e *n*) e comunicando il risultato.
Il programma che abbiamo appena visto può presentare un difetto. Supponiamo di essere di fronte a un terminale. Il calcolatore che sta eseguendo questo algoritmo è collegato al nostro terminale e si ferma per chiederci il valore da assegnare alla variabile *n*. Ovviamente il programma è concepito per ricevere un *numero*. Ma la tastiera del terminale ha anche delle lettere, e noi potremmo decidere di fornire in ingresso l'espressione 'Mi sto'. Dopo ciò, il calcolatore si ferma per chiederci il valore da assegnare a *m*. Proseguendo nella nostra burla, battiamo sulla tastiera 'divertendo'. A questo punto il calcolatore raggiunge il quarto passo: deve calcolare la somma di 'Mi sto' e di 'divertendo'. Ma qual è la somma, in senso aritmetico, di 'Mi sto' e di 'divertendo'? Non esiste una risposta chiara a questa domanda. Se provassimo davvero a fornire questi dati di ingresso a un calcolatore programmato per eseguire il diagramma di flusso, potremmo ottenere diversi tipi di reazioni. Il calcolatore potrebbe fermarsi del tutto, oppure potrebbe emettere un messaggio di errore, o addirittura potrebbe fornire in uscita qualche espressione strana, che esso considera essere la somma delle due espressioni che gli abbiamo fornito. Il calcolatore non ne risulterebbe comunque danneggiato, poiché è praticamente impossibile rompere un calcolatore con azioni di questo tipo.
Per migliorare il programma, impedendo che qualche pasticcione lo usi scorrettamente, possiamo modificarlo nella maniera seguente (Figura 2.8).
Se a *n* o a *m* non viene assegnato un valore numerico, un calcolatore che stia eseguendo il programma ce lo fa notare visualizzando il messaggio "Non è un numero!", poi si ferma. Se invece sia *n* che *m* sono numeri, viene correttamente fornita in uscita la loro somma. Si noti come abbiamo usato il test (nel rombo) per gestire eventualità indesiderate; ovviamente, i test possono essere usati anche per gestire eventualità diverse, ma tutte ugualmente ammissibili.
A questo punto si dovrebbe avere un'idea di che cosa sia un algoritmo e di come si possa usare un diagramma di flusso per rappresentarne uno. È difficile fornire un criterio per determinare se un diagramma di flusso rappresenta effettivamente un algoritmo. Intuitivamente, si può dire che un diagramma di flusso non rappresenta un algoritmo se è possibile individuare, in esso, almeno un flusso di controllo che non raggiunga mai un nodo 'STOP'.
Non possiamo fornire qui tutte le regole per determinare quando un diagramma di flusso rappresenta un algoritmo; possiamo però dare alcune indicazioni per la costruzione di un diagramma di flusso corretto:

1. Ogni rettangolo esecutivo deve avere una e una sola freccia di uscita.
2. Ogni rombo di test deve avere due e due sole frecce di uscita, una marcata con TRUE (oppure con "sì") e l'altra marcata con FALSE (oppure con "no").

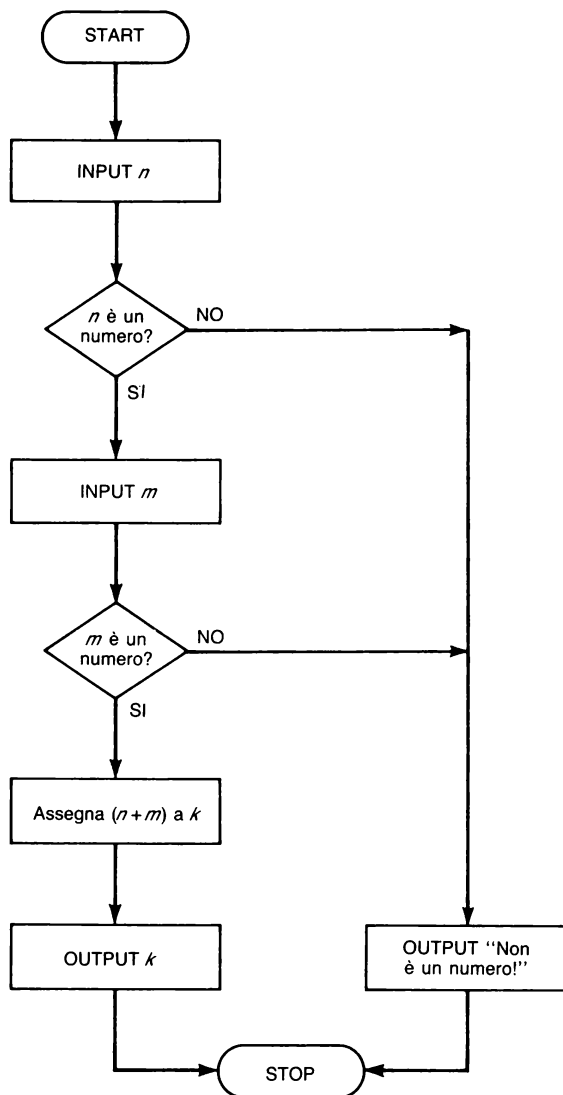



Figura 2.8 Diagramma di flusso per sommare due numeri qualsiasi, con messaggi d'errore

3. Tutti i nodi, esclusi START e STOP, devono avere almeno due frecce che li collegano ad altri nodi.
4. I collegamenti fra i nodi sono a senso unico: il flusso di controllo da un nodo a un altro può andare in una sola direzione.

Purtroppo la rappresentazione mediante diagrammi di flusso presenta delle limitazioni. Quando il compito da eseguire è complesso e richiede un algoritmo lungo, la rappresentazione dell'algoritmo mediante diagramma di flusso può occupare molte pagine e risultare pressoché illeggibile. I diagrammi di flusso sono utili e chiarificatori solo quando sono relativamente semplici. Un altro difetto dei diagrammi di flusso è che non assomigliano molto ai prodotti finali della descrizione degli algoritmi, cioè ai programmi scritti in un linguaggio di programmazione dei calcolatori. Un programma per calcolatore è infatti una sequenza di istruzioni priva di cornici, ramificazioni e frecce di collegamento.

2.6 UNO PSEUDOLINGUAGGIO DI PROGRAMMAZIONE

In alternativa ai diagrammi di flusso, per descrivere algoritmi si può usare uno *pseudolinguaggio di programmazione*. Un algoritmo rappresentato per mezzo di uno pseudolinguaggio di programmazione è molto più simile a un programma per calcolatore. Useremo dunque questo metodo, se appena un algoritmo risulterà lungo o complesso. La descrizione di un algoritmo tramite uno pseudolinguaggio di programmazione, anziché tramite un diagramma di flusso, permette una più semplice conversione dell'algoritmo stesso in un effettivo programma per calcolatore, oppure in una sequenza perfettamente comprensibile di istruzioni espresse in italiano corrente.
Lo pseudolinguaggio di programmazione che useremo in questo libro ha le seguenti caratteristiche:

1. Ogni algoritmo è descritto da una sequenza numerata di istruzioni.
2. La forma di tali istruzioni è limitata a pochi modelli fondamentali.
3. Le istruzioni subordinate o dipendenti sono scritte con margine sinistro rientrato.
4. Porzioni ampie di un algoritmo, le quali svolgano un compito secondario e ben circoscritto, vengono separate e rappresentate da un nome.

Quest'ultima caratteristica è spesso chiamata *progettazione modulare* e costituisce il fondamento di un buono stile di programmazione. In pratica, la progettazione modulare richiede dapprima un'analisi discendente del problema, poi una suddivisione dell'algoritmo in *sottoprogrammi*, ciascuno dei quali isola un sottoproblema ben preciso e definito. Nel nostro pseudolinguaggio di programmazione useremo nomi a tutte maiuscole per individuare gli algoritmi e i sottoprogrammi che li costituiscono.

Il nostro pseudolinguaggio di programmazione permette solo due tipi fondamentali di istruzioni: le *istruzioni esecutive* (o *operazioni*) e le *istruzioni di controllo*. Un'istruzione esecutiva indica un'operazione o un'azione che l'esecutore dell'algoritmo deve effettuare. Le istruzioni di controllo indicano quante volte, o sotto quali condizioni, o in che ordine devono essere eseguite delle operazioni. Tali istruzioni controllano l'esecuzione delle operazioni.
Una parte importante di molte istruzioni di controllo è la *condizione*. Una condizione è un enunciato che può valere TRUE e FALSE, ma non può essere, in sé, un'istruzione. Ad esempio, '4 > 2' è un enunciato il cui valore di verità è TRUE, e quindi potrebbe essere una condizione. Non ha invece senso ordinare a qualcuno di rendere 4 maggiore di 2, quindi non si tratta di un'operazione. In una istruzione condizionale di controllo, la condizione è seguita a sua volta da un'istruzione (esecutiva o di controllo), o da una sequenza di istruzioni.
Si consideri il seguente esempio di istruzione di controllo:

SE il saldo del vostro conto è minore di £ 100.000,
ALLORA pagate £ 5.000 per spese bancarie.

L'elemento di controllo in questa istruzione è 'SE <condizione>'. In questo esempio, la condizione è 'il saldo del vostro conto è minore di £ 100.000'. La parte esecutiva dell'istruzione contiene una sola operazione: 'pagate £ 5.000 per spese bancarie'. Dunque, in un'istruzione di controllo, la condizione indica *quando* si deve fare qualcosa, mentre la parte esecutiva indica *che cosa* si deve fare.

ISTRUZIONI ESECUTIVE

Le operazioni contenute in un algoritmo devono essere molto semplici, e comunque tali che ci si possa ragionevolmente aspettare che l'esecutore (uomo o calcolatore) sia in grado di capirle, senza aver bisogno di ulteriori istruzioni. Le operazioni più complesse devono essere descritte tramite una sequenza di diverse operazioni più semplici. Poiché i calcolatori, al contrario dei robot, non fanno quasi altro che chiedere e fornire informazioni agli utenti umani, due operazioni comuni sono:

INPUT ...

e

OUTPUT ...

L'istruzione 'INPUT ...' impone all'esecutore dell'algoritmo di ottenere un'informazione dall'ambiente, cercandola nelle proprie registrazioni o chiedendola all'utente. L'istruzione 'OUTPUT ...' impone all'esecutore dell'algoritmo di comuni-

care un'informazione (ad esempio, stampandola). Poiché queste due operazioni richiedono all'esecutore dell'algoritmo di interagire con un ambiente esterno, cioè con il mondo esterno al calcolatore, sono spesso chiamate operazioni esterne, o, più propriamente, operazioni di *ingresso-uscita*.
Ci sono però anche delle operazioni, dette operazioni interne, che l'algoritmo usa per scopi suoi propri. Fra queste, è fondamentale l'operazione di *assegnamento*, che consiste nell'associare un'informazione a un nome. Tale informazione può poi essere recuperata più avanti nell'algoritmo, tramite il suo nome. Una *variabile* è appunto un nome associato (o associabile) a un'informazione. Degli esempi di assegnamento possono essere:

Sia *n* uguale a 5.
Assegna a *n* il vecchio valore di *n* incrementato di 1.
Sia QUESTASTRINGA la stringa 'La logica è semplice'.

Ciascuna di queste istruzioni assegna un'informazione a un nome (qui alla lettera '*n*' o alla parola 'QUESTASTRINGA'). Tale informazione può essere un numero, un valore di verità, o una sequenza di caratteri (detta *stringa*). Di solito, i linguaggi di programmazione dei calcolatori impongono restrizioni sui tipi di informazione assegnabili alle variabili. La classe di informazioni assegnabili a una variabile è detta *tipo di dato* (o *tipo* della variabile). In questo libro non ci occuperemo del tipo di informazioni associabili ai nomi, e permetteremo quindi che qualsiasi nome si riferisca a qualsiasi tipo di informazione. Non imporremo neppure restrizioni sulla struttura dei nomi usati come variabili, anche se spesso sceglieremo un nome, come 'QUESTASTRINGA', che ci aiuti a ricordare quale informazione è associata alla variabile. Nel nostro pseudolinguaggio, l'assegnamento avrà la forma:

LET <nome della variabile> = <informazione da assegnare>

ISTRUZIONI DI CONTROLLO

In un algoritmo, le istruzioni di controllo sono tanto importanti quanto quelle esecutive, poiché sapere quando fare qualcosa, e quanto spesso, è tanto importante quanto sapere che cosa fare.
Una caratteristica importante del nostro pseudolinguaggio di programmazione è la seguente: salvo se diversamente indicato, le istruzioni devono essere eseguite nell'ordine in cui compaiono.
Ad esempio, se abbiamo l'algoritmo

1. LET *n* = 5.
2. LET nuovo valore di *n* = vecchio valore di *n* + 1.
3. OUTPUT *n*.
4. STOP.

esso verrà eseguito nel modo seguente. Dopo il passo 1, la variabile n contiene l'informazione '5'. Dopo il passo 2, n assume il proprio valore precedente, incrementato di 1, cioè contiene l'informazione '6'. Infine, il valore di n viene fornito in uscita (passo 3). Poiché, a questo punto dell'algoritmo, n contiene l'informazione '6', il risultato finale osservabile dell'algoritmo consiste semplicemente nel fornire '6' in uscita.
Un'istruzione di controllo fondamentale in un algoritmo è, come abbiamo visto, l'istruzione condizionale 'SE ... ALLORA ...'. Nel nostro pseudolinguaggio di programmazione, come in molti effettivi linguaggi di programmazione dei calcolatori, essa avrà la forma

IF <condizione>
  THEN <istruzione>

ove appunto 'IF' = 'se' e 'THEN' = 'allora'.
Consideriamo un esempio di algoritmo che usi un'istruzione condizionale:

1. LET n = 7.
2. LET m = 5 + 2.
3. IF il valore di n è uguale al valore di m,
   THEN (a) OUTPUT "n e m hanno lo stesso valore".
4. STOP.

Poiché, dopo l'esecuzione dei primi due passi, la condizione al passo 3 vale TRUE, viene effettuata l'operazione indicata al passo 3(a). Viene perciò stampata o visualizzata la frase "n e m hanno lo stesso valore". Se la condizione avesse avuto valore di verità FALSE, tale frase non sarebbe stata visualizzata.
Altre utili istruzioni di controllo sono:

FOR <sequenza di valori>
  <istruzioni>
WHILE <condizione>
  <istruzioni>
GO TO <passo dell'algoritmo>
STOP

Il primo tipo di istruzione di controllo si chiama *ciclo FOR* ('FOR' significa 'per'). Esso comporta che le istruzioni indicate vengano ripetute il numero di volte specificato dalla sequenza di valori. Vediamo un esempio di applicazione del ciclo FOR:

1. INPUT una frase.
2. FOR ogni carattere della frase
   (a) IF il carattere è una 'e'
      THEN OUTPUT "Lettera 'e'".
3. STOP

Se la frase fornita in ingresso all'algoritmo al passo 1 è

Questa è una frase.

al passo 2 dell'algoritmo viene esaminato ogni carattere della frase (lettere, spazi bianchi e segni di interpunzione). Ogni volta che si incontra la lettera 'e', il passo 2(a) fornisce in uscita il messaggio "Lettera 'e'". Il successivo passo 3 viene eseguito solo dopo che ogni carattere della frase è stato esaminato. In altre parole, il passo 2(a) viene ripetuto una volta per ogni carattere della frase d'ingresso. Solo dopo ciò il controllo si sposta al passo 3.
Il secondo tipo di istruzione di controllo si chiama *ciclo WHILE* ('WHILE' qui corrisponde a 'finché'). Esso comporta che le istruzioni marginate vengano ripetute finché la condizione vale TRUE. Non appena la condizione assume il valore di verità FALSE, il controllo passa alla successiva istruzione non marginata. Sia il ciclo FOR che il ciclo WHILE indicano, in modi diversi, quante volte devono essere ripetute delle istruzioni (distinte dalle altre mediante la marginazione).
L'istruzione GO TO (che significa "vai a", ossia "salta al passo ...") indica che si deve passare ad eseguire un altro passo dell'algoritmo. Poiché è abbastanza comune che ogni passo di un algoritmo sia contrassegnato in qualche modo, l'istruzione GO TO spesso si presenta in una forma del tipo 'GO TO passo 4'.
Le istruzioni GO TO sono molto malviste dai programmatori più accorti, perché generano confusione: quando si legge un algoritmo, si è costretti a saltare avanti e indietro fra i vari passi che lo compongono. Di conseguenza, le useremo raramente. Il loro uso potrebbe anche essere completamente eliminato, ma in parecchi degli algoritmi, che descriveremo in modo informale, è più semplice e più chiaro usare i GO TO, piuttosto che evitarli.
L'istruzione di controllo dal significato più ovvio è STOP, che indica che si è giunti al termine dell'algoritmo.
Si consideri il seguente algoritmo, descritto con il nostro pseudolinguaggio di programmazione:

1. INPUT il valore di n.
2. INPUT il valore di m.
3. IF n non è un numero, oppure m non è un numero,
   THEN (a) OUTPUT "I valori di ingresso non sono entrambi numerici".
   (b) STOP.
4. IF m vale 0
   THEN (a) OUTPUT "Impossibile dividere per 0".
   (b) STOP.
5. LET $k = n/m$.
6. OUTPUT k.
7. STOP.

Questo algoritmo è stato progettato per il semplice scopo di dividere un numero per un altro. È stato però anche progettato per evitare le difficoltà causate dal

mancato assegnamento di valori numerici alle variabili *n* e *m* (passo 3) e dalla divisione per 0 (passo 4). Questo algoritmo presenta diverse caratteristiche importanti.
Innanzitutto, l'algoritmo deve essere eseguito passo per passo, saltando i passi marginati se una condizione vale FALSE, finché si raggiunge uno STOP. Si fa esattamente quanto indicato da ciascun passo. Quando si giunge a un'istruzione condizionale, se la condizione vale TRUE si eseguono le corrispondenti istruzioni marginate; se invece la condizione vale FALSE, tali istruzioni non vengono eseguite, e si salta al prossimo passo non marginato.
Inoltre, accade spesso che si debbano eseguire più istruzioni se una condizione vale TRUE. In questi casi, tali istruzioni vengono marginate rispetto alla condizione da cui dipendono. Questa disposizione è stata usata nei passi 3 e 4.

2.7 CONCLUSIONI

I calcolatori possono essere usati per risolvere problemi di logica, ad esempio per definire se una data argomentazione è valida, oppure per fornire prove che dimostrino perché un'argomentazione è valida. Per indicare a un calcolatore come risolvere tali problemi logici, è fondamentale il concetto di *algoritmo*: una sequenza di istruzioni per l'esecuzione di un compito, tale che, fornita ogni informazione richiesta, il compito possa essere condotto a termine in un numero finito di passi; tali istruzioni devono inoltre essere del tutto meccaniche e prive di ambiguità, nel senso che l'esecutore (umano o artificiale) delle istruzioni non deve aver bisogno di creatività o di informazioni aggiuntive. Un *programma* è una rappresentazione di un algoritmo, che ne permette la corretta esecuzione da parte di uno specifico calcolatore.
Rappresenteremo gli algoritmi o mediante *diagrammi di flusso,* o mediante uno *pseudolinguaggio di programmazione*. Rispetto al metodo dei diagrammi di flusso, il nostro pseudolinguaggio di programmazione permette di rappresentare un algoritmo in una forma molto più simile a quella di un effettivo programma per calcolatore.
Sia nella rappresentazione con diagrammi di flusso, sia nello pseudolinguaggio di programmazione, ogni istruzione di un algoritmo può essere di uno dei seguenti tipi:

Istruzioni esecutive

1. Esterne
   a. INPUT
   b. OUTPUT
2. Interne
   a. Assegnamento: LET <variabile> = <informazione>

Istruzioni di controllo

1. Controllo incondizionato
   a. STOP
   b. GO TO
2. Controllo condizionato
   a. IF <condizione>
      THEN <istruzioni>
   b. FOR <sequenza di valori>
      <istruzioni>
   c. WHILE <condizione>
      <istruzioni>

2.8 ESERCIZI

A. Un importante criterio per determinare se un calcolatore è in grado di pensare è il cosiddetto *test di Turing*, proposto dal matematico e informatico inglese Alan Turing (vedi Turing, 1950, in Anderson, 1964). Una sua versione semplificata è la seguente. Si supponga di essere di fronte a un terminale, o ad un altro dispositivo di comunicazione, e di non sapere chi o che cosa ci sia all'altro capo della linea di comunicazione. Tramite il terminale, si comincia una conversazione, ponendo e ricevendo domande e risposte su se stessi, sulle proprie opinioni e sul mondo.
Si deve anche giocare, chiedere consigli, e in generale fare tutto ciò che si fa quando si comunica con altri esseri umani. Se lo svolgimento del dialogo porta alla convinzione che si sta comunicando con un essere umano, allora si deve ammettere che qualunque cosa si trovi all'altro capo della linea, anche se è un calcolatore, è in grado di pensare. In altre parole, qualunque cosa che sappia conversare come un essere umano pensante deve essere considerata capace di pensare.
Rispondere alle seguenti domande sul test di Turing:
   1. È possibile che, comunicando con un altro essere umano pensante, non ci rendiamo conto che egli stia effettivamente pensando? Ciò può accadere, ad esempio, se la persona all'altro capo della linea di comunicazione non sa usare la tastiera. Ovviamente, ciò non significa necessariamente che tale persona non sia capace di pensare. Quali altri possibili motivi potrebbero impedire a un essere umano pensante di fornirci prove della sua capacità di pensare?
   2. Alla luce della prima domanda, considerare queste due affermazioni:
      a. Se una persona o un calcolatore supera il test di Turing, allora è capace di pensare.
      b. Se una persona o un calcolatore non supera il test di Turing, allora non è capace di pensare.

Queste affermazioni sono entrambe ragionevoli? (Il programma ELIZA è considerato un controesempio dell'affermazione (a); vedi Weizenbaum, 1976.)

3. Supponendo di essere a un terminale e di sottoporre al test di Turing un essere umano o artificiale, costruire tre domande e risposte considerate sufficienti per determinare se l'essere sottoposto al test è capace di pensare. Suggerimento: almeno una delle domande deve richiedere *ragionamento*, e non semplice conoscenza.

B. Il seguente algoritmo, scritto nel nostro pseudolinguaggio di programmazione, propone un breve questionario geografico e valuta le risposte.

```
 1. LET A = 0.
 2. Rispondere TRUE o FALSE: "La capitale della Grecia è Atene".
 3. LET V (GR) = risposta.
 4. Rispondere TRUE o FALSE: "La capitale della Svezia è Oslo".
 5. LET V (SV) = risposta.
 6. Rispondere TRUE o FALSE: "La capitale dell'Albania è Tirana".
 7. LET V (AL) = risposta.
 8. IF V (GR) = TRUE
       THEN LET A = valore precedente di A + 1.
 9. IF V (SV) non è TRUE
       THEN LET A = valore precedente di A + 1.
10. IF V (AL) non è FALSE
       THEN LET A = valore precedente di A + 1.
11. IF A = 3
       THEN OUTPUT "Risposte tutte giuste".
12. IF A = 2
       THEN OUTPUT "Riprova: sarai più fortunato".
13. IF A < 2
       THEN OUTPUT "Pietoso!".
14. STOP.
```

a. Rispondere al questionario. Quali sono le risposte fornite ai passi 3, 5 e 7? Che cosa fa il programma dopo il passo 10?

b. Se si risponde correttamente a tutte le domande, il programma che cosa fornisce in uscita?

c. Si supponga che le risposte ai passi 3, 5 e 7 siano:

   TRUE
   NON SO
   FALSE

   Quanto vale A al passo 4? E dopo il passo 8? E dopo il passo 9? E dopo il passo 10? Qual è il giudizio emesso dal programma?

d. Rispondere alle stesse domande di (c), nel caso in cui le risposte al questionario siano:

   FALSE
   NON MI RICORDO

CHE QUESTIONARIO CRETINO!

e. Riscrivere i passi 9 e 10 in modo che il giudizio del programma sulle risposte sia sempre corretto.

C. Considerare il seguente algoritmo, che propone un questionario sui pianeti e ne valuta le risposte.

1. Rispondere: "Qual è il pianeta successivo alla Terra?".
2. LET P4 = risposta.
3. Rispondere: "Qual è il pianeta più vicino al Sole?".
4. LET P1 = risposta.
5. Rispondere: "Qual è il pianeta più lontano dal Sole?".
6. LET P9 = risposta.
7. Rispondere: "Qual è il pianeta più grande?".
8. LET P5 = risposta.
9. LET A = 100.
10. IF P5 non è 'GIOVE'
    THEN LET A = valore precedente di A − 25.
11. IF P9 non è 'PLUTONE'
    THEN LET A = valore precedente di A − 25.
12. IF P1 = 'MERCURIO'
    THEN GO TO passo 14.
13. LET A = valore precedente di A − 25.
14. IF P4 = 'MARTE'
    THEN GO TO passo 16.
15. LET A = valore precedente di A − 25.
16. OUTPUT "Il voto che avete meritato è: ".
17. OUTPUT A.
18. STOP.

a. Se ai passi 2, 4, 6 e 8 si risponde rispettivamente MERCURIO, MERCURIO, PLUTONE e GIOVE, che voto si ottiene?

b. Se ai passi 2, 4, 6 e 8 si risponde rispettivamente VENERE, SATURNO, PLUTONE e GIOVE, qual è il valore di A dopo il passo 10? E dopo il passo 11? E dopo il passo 13?

D. Il raffinamento del passo 2 del primo algoritmo per la somma del paragrafo 2.4 è incompleto, perché non indica quando ci si deve fermare. Riscrivere questo passo in modo che l'esecutore dell'algoritmo sappia quando arrestarsi.

E. Si scriva un breve algoritmo per l'esecuzione di qualche semplice compito, si provi a farlo eseguire a un'altra persona e si veda se essa è in grado di eseguirlo senza ricorrere alla creatività o all'immaginazione.

F. La logica simbolica tratta espressioni costituite da lettere ('A', 'B', ...) e da altri simboli, come ')', '(', 'v' e '~'. Considerare i seguenti compiti, per la cui esecuzione si può scrivere un algoritmo.

1. Talora è importante sapere *quante volte* un dato simbolo compare in una stringa di simboli. Ad esempio, è facile vedere che nella stringa
   ASFAHDA$%123

la lettera 'A' compare tre volte. Scrivere un algoritmo per determinare quante volte compare la lettera 'A' in una stringa d'ingresso.

2. Un'altra funzione importante, che useremo in seguito, è il calcolo del *numero di simboli distinti* che compaiono in una stringa. Ad esempio, nella stringa

   ASDF%$121AD$%1

   compaiono otto simboli distinti:A, S, D, F, %, $, 1, 2. Scrivere un algoritmo che riceva in ingresso una stringa e fornisca in uscita il numero di simboli distinti di tale stringa.

G. Considerare il seguente algoritmo per comprare del latte:

1. Andare al supermercato.
2. IF hanno il latte
   THEN comprarlo.
3. IF sono senza latte
   THEN (a) andare a un altro supermercato.
   (b) GO TO passo 2.
4. Tornare a casa.
5. STOP.

Rispondere alle seguenti domande:

a. Disegnare il diagramma di flusso di questo algoritmo.
b. Che cosa succede se c'è un numero indefinito di supermercati, tutti privi di latte?
c. Correggere l'algoritmo in modo che tratti il caso (b).
d. Disegnare il diagramma di flusso di tale versione corretta dell'algoritmo.

3

Logica enunciativa: i connettivi ‘non’, ‘e’ ed ‘o’

Una delle branche più importanti della logica è la logica degli enunciati, detta appunto *logica enunciativa*. La logica enunciativa esamina le argomentazioni considerandone come parti elementari gli *enunciati dichiarativi*.
Considereremo un enunciato come una sequenza, o stringa, di simboli che esprime una *proposizione* (la logica enunciativa si chiama perciò anche *logica proposizionale*). Due o più enunciati distinti possono esprimere la stessa proposizione. Si considerino le stringhe seguenti:

La Terra è rotonda.
Rotonda è la terra.
La Terre est ronde.

Ogni riga contiene una stringa di simboli (in questo caso, lettere, spazi bianchi e punti). Il primo enunciato, per esempio, è costituito dalla sequenza di simboli ‘L’, ‘a’, ‘ ’ (spazio bianco), ‘T’, eccetera.
Le tre righe sopra riportate sono tre stringhe distinte, o per l’ordine dei simboli, o per il diverso insieme di simboli usato. Tutte, comunque, esprimono all’incirca lo stesso pensiero, o proposizione. È evidente che certe stringhe non rappresentano enunciati, perché non esprimono proposizioni in nessun linguaggio noto. Ad esempio, le stringhe seguenti non esprimono proposizioni, e quindi non sono enunciati:

Egl%! Oglrn.%
Glub af noc.

Si consideri, come esempio su cui lavorare, la proposizione che afferma che Andrea ha spedito un libro a Marco.
In italiano tale proposizione può essere espressa dall’enunciato:

1. Andrea ha spedito un libro a Marco.

Esistono delle regole grammaticali che permettono di trasformare questo enunciato in altri enunciati che esprimono la stessa proposizione. Ad esempio, l'enunciato

2. Andrea ha spedito a Marco un libro.

esprime ancora la stessa proposizione dell'enunciato (1). Si noti che (1) e (2) sono enunciati distinti, poiché sono due stringhe distinte di simboli.

3.1 NEGAZIONE

Potrebbe però essere falso che Andrea abbia spedito un libro a Marco. Ci sono diversi modi in cui questa proposizione potrebbe non essere vera:

Non è *Andrea* che ha spedito il libro.
Andrea non l'ha *ancora* spedito.
Non l'ha *spedito*, bensì lo ha consegnato personalmente.
Andrea ha spedito un libro, ma non a *Marco*.
Quello che Andrea ha spedito non è un *libro*.

Anche una qualsiasi combinazione di questi fatti renderebbe falsa la proposizione originaria. Ognuno degli esempi di questa lista potrebbe essere formulato in diversi modi. Poiché però, ai fini della logica enunciativa, interessano solo la verità o la falsità delle proposizioni, basterà la proposizione generica e aspecifica che afferma *che non si dà il caso che Andrea abbia spedito un libro a Marco*. Questa proposizione nega la proposizione originaria, e può venire convenzionalmente espressa appunto mediante l'enunciato "Non si dà il caso che Andrea ha spedito un libro a Marco".
Per costruire un enunciato che neghi una proposizione basta dunque prendere un enunciato che esprima la proposizione originaria e prefiggere la stringa 'Non si dà il caso che' alla particolare stringa di simboli che costituisce tale enunciato. L'enunciato che così si ottiene è la *negazione* dell'enunciato originario.
Per evitare di riscrivere più volte gli enunciati e per rappresentarli in forma più semplice, seguendo la convenzione introdotta nel Capitolo 1 useremo le lettere maiuscole in neretto '**P**', '**Q**', '**R**', ..., '**Z**' per indicare una qualsiasi stringa che sia un enunciato. Diremo perciò che il modo convenzionale per formare la negazione di un enunciato **P** consiste nello scrivere:

Non si dà il caso che **P**.

Possiamo semplificare ulteriormente le cose sostituendo la stringa 'Non si dà il caso che' con un unico simbolo: ¬. La negazione di un enunciato **P** sarà perciò rappresentata da:

¬**P**

Alcuni esempi concreti sono illustrati nella Tabella 3.1.

Tabella 3.1 Negazioni di enunciati

Enunciato	**Negazione corrente**	**Negazione convenzionale**	**Negazione simbolica**
Fido abbaia.	Fido non abbaia.	Non si dà il caso che Fido abbaia.	¬(Fido abbaia.)
$2+2 = 4$	$2+2 \neq 4$	Non si dà il caso che $2+2 = 4$	$\neg(2+2 = 4)$
$7<10$	7 non è minore di 10	Non si dà il caso che $7<10$	$\neg(7<10)$

Le proposizioni possono essere vere oppure false. Un enunciato ha valore di verità TRUE se esprime una proposizione vera, FALSE se esprime una proposizione falsa. Secondo la convenzione definita nel Capitolo 1, useremo le parole a tutte maiuscole 'TRUE' e 'FALSE' per indicare i due possibili valori di verità degli enunciati. È opportuno indicare gli enunciati e i loro valori di verità con simboli distinti, anche se correlati. Continueremo ad usare le lettere maiuscole per indicare gli enunciati, e la notazione 'V( )' per rappresentare il valore di verità di un enunciato. Ad esempio:

C = 'La Terra è piatta.'
V(C) = FALSE

Come possiamo determinare, in generale, quando vale TRUE la negazione di un enunciato? Ad esempio, quando vale TRUE il seguente enunciato?

Non si dà il caso che Andrea ha spedito un libro a Marco.

La risposta dovrebbe essere intuitiva: la negazione di un enunciato vale TRUE esattamente quando l'enunciato stesso vale FALSE. Nel nostro esempio,

Non si dà il caso che Andrea ha spedito un libro a Marco.

vale TRUE esattamente quando

Andrea ha spedito un libro a Marco.

vale FALSE. Inoltre, la negazione di un enunciato vale FALSE esattamente quando l'enunciato originario vale TRUE. Il valore di verità di una negazione dipende dunque solo dal valore di verità dell'enunciato originario.

LA FUNZIONE DI VERITÀ FNEG

La relazione di dipendenza fra un enunciato e la sua negazione può essere definita in modo più chiaro e preciso dicendo che il valore di verità della negazione di un enunciato è una funzione del valore di verità dell'enunciato originario. Una *funzione* produce una singola uscita per un dato ingresso (o insieme di ingressi). Diremo che la negazione è una *funzione di verità*, perché fornisce in uscita un valore di verità (il valore di verità della negazione) a partire da un ingresso costituito dal valore di verità dell'enunciato originario.

Come esempio del concetto di funzione, si consideri la distanza percorsa da un oggetto in movimento: essa dipende dalla velocità dell'oggetto e dalla durata dell'intervallo di tempo durante il quale l'oggetto si è spostato a tale velocità. Diremo perciò che la distanza percorsa è funzione della velocità e del tempo. Analogamente, il valore di una semplice somma aritmetica ($m + n$) dipende dai valori degli addendi.

Matematici e scienziati hanno sviluppato una notazione assai diffusa per esprimere queste relazioni di dipendenza, o relazioni funzionali. Potremmo ad esempio scrivere

Distanza = FDIS(Velocità, Tempo)

per indicare che la distanza percorsa da un oggetto in movimento è funzione sia della sua velocità, sia del tempo durante il quale esso si è mosso a tale velocità. Analogamente, potremmo scrivere

Somma = FSOM(Addendo 1, Addendo 2)

per indicare che, come è ovvio, il valore di una somma è funzione di entrambi i numeri che vengono sommati.

I nomi che abbiamo usato per queste funzioni iniziano per 'F': FDIS e FSOM. Ciò serve ad indicare che si tratta di funzioni. Le lettere successive servono per distinguere fra loro le diverse funzioni, ma è anche opportuno che vengano scelte in modo da indicare il significato delle singole funzioni. In questi due casi, FDIS e FSOM appaiono due scelte ragionevoli per indicare la funzione "distanza percorsa" e la funzione "somma".

Anche per descrivere le funzioni di verità, come la negazione, adotteremo queste convenzioni. I nomi delle funzioni di verità inizieranno per 'F', mentre le lettere seguenti indicheranno il significato delle singole funzioni. Useremo pertanto FNEG come nome della funzione di negazione. Perciò scrivendo

V(¬**P**) = FNEG(V(**P**))

esprimiamo semplicemente il fatto che il valore di verità di una negazione ¬**P** è una funzione (FNEG) del valore di verità di **P**.

Una funzione può essere descritta in due modi. Il primo modo consiste nell'elencarne tutti i possibili ingressi e le corrispondenti uscite. La funzione viene così de-

scritta *estensionalmente*. Ad esempio, le seguenti tabelle descrivono parzialmente le funzioni FDIS e FSOM in modo estensionale:

Ingressi		**Uscita**
Velocità (km/h)	**Tempo (ore)**	**FDIS(Velocità, Tempo)**
15	1	15
2	10	20
30	3	90

eccetera

Ingressi		**Uscita**
m	n	**FSOM(** m, n**)**
0	0	0
1	0	1
3	2	5

eccetera

È evidente che questo modo di descrivere una funzione risulta spesso molto gravoso. Nei due esempi sopra riportati, l'elenco dovrebbe proseguire indefinitamente, per poter descrivere completamente le funzioni.
Una funzione può però essere descritta anche *intensionalmente*, fornendo un algoritmo che determini l'uscita della funzione a partire dai suoi ingressi. L'algoritmo costituisce un metodo (una "ricetta") per generare l'uscita di una funzione a partire dai suoi ingressi. Se gli ingressi e le uscite di un algoritmo corrispondono agli ingressi e alle uscite di una funzione, si dice che l'algoritmo *calcola* tale funzione. Ciò non significa però che l'algoritmo e la funzione siano la stessa cosa. Ad esempio, un algoritmo che calcola FSOM è

1. INPUT m.
2. INPUT n.
3. OUTPUT $m + n$.
4. STOP.

mentre un algoritmo che calcola FDIS è

1. INPUT velocità.
2. INPUT tempo.
3. OUTPUT velocità × tempo.
4. STOP.

Fornire un algoritmo che calcola una funzione è spesso più facile che elencare

ogni possibile ingresso e uscita. In effetti, fornire un algoritmo è spesso l'unico modo per descrivere completamente una funzione. Una funzione è completamente descritta quando la sua uscita è specificata per ognuno dei possibili ingressi che essa può avere. Naturalmente, possono esistere più algoritmi distinti che calcolano la stessa funzione.

TAVOLE DI VERITÀ E ALGORITMI PER IL CALCOLO DI FNEG

La funzione di verità FNEG può essere descritta, in modo quasi ugualmente semplice, sia mediante un elenco di ingressi e uscite, sia mediante un algoritmo. Ciò è possibile in quanto, mentre le variabili numeriche possono assumere un numero infinito di valori (1, 2, 3, ...), i valori di verità possono essere solo due: TRUE e FALSE.
La descrizione estensionale completa della funzione FNEG è:

Ingresso V(**P**)	**Uscita** FNEG(V(**P**))
FALSE	TRUE
TRUE	FALSE

La descrizione estensionale di una funzione di verità, come quella sopra riportata, si dice *tavola di verità*. Nelle tavole di verità riporteremo sempre gli ingressi a sinistra della riga verticale e l'uscita a destra di essa, e non sempre metteremo le intestazioni "Ingresso" e "Uscita".
Possiamo descrivere la funzione FNEG anche fornendo un algoritmo che produca gli stessi risultati illustrati dalla tavola di verità. Potremmo dire, in modo informale, che FNEG "inverte" il valore di verità del proprio ingresso. Forse però è inutile ricorrere a questa espressione informale, giacché l'idea di "invertire" un valore di verità, benché intuitivamente chiara, non costituisce un concetto applicato rigorosamente dalle persone o dai calcolatori.
È inoltre possibile disegnare un diagramma di flusso che rappresenta un algoritmo per calcolare la funzione FNEG di un arbitrario valore di verità di un enunciato **P** ricevuto in ingresso (Figura 3.1).
In alternativa, si può descrivere la funzione FNEG mediante un algoritmo scritto nel nostro pseudolinguaggio di programmazione:

```
1. INPUT V(P).
2. IF V(P) = TRUE
      THEN OUTPUT "FALSE."
3. IF V(P) = FALSE
      THEN OUTPUT "TRUE."
4. STOP.
```

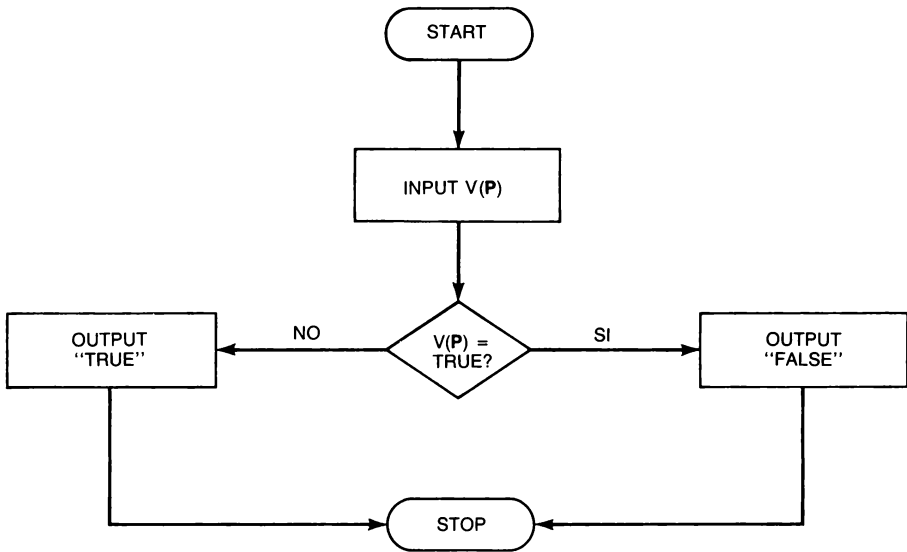



Figura 3.1 Diagramma di flusso per il calcolo della funzione FNEG

Finora abbiamo detto che il valore di verità di un enunciato può essere o TRUE o FALSE. Ricordando l'associazione del numero 1 con TRUE e del numero 0 con FALSE, possiamo realizzare un algoritmo molto conciso per descrivere la funzione FNEG. Notiamo innanzitutto che, usando i numeri 0 e 1, la tavola di verità di FNEG diventa

V(**P**)	FNEG(V(**P**))
0	1
1	0

Poiché adesso operiamo su numeri, per trovare un algoritmo che descriva FNEG potremmo cercare qualche semplice operazione aritmetica che si comporti come tale funzione.
Ecco un semplice algoritmo aritmetico adatto a questo scopo:

1. INPUT V(**P**).
2. OUTPUT 1 − V(**P**).
3. STOP.

Si può dimostrare molto facilmente che questa operazione aritmetica si comporta esattamente come FNEG. Se V(**P**) = 1 (cioè se **P** vale TRUE), allora 1 − V(**P**)

= 0. Se V(**P**) = 0 (cioè se **P** vale FALSE), allora 1 − V(**P**) = 1. In altre parole, l'algoritmo costituito dalla semplice operazione 1 − V(**P**) "inverte" i valori di verità, proprio come fa FNEG.

DOPPIA NEGAZIONE

Abbiamo visto che fra la negazione di un enunciato e l'enunciato originario intercorre una relazione funzionale: il valore di verità della negazione dipende esclusivamente dal valore di verità dell'enunciato originario. Si ha cioè:

V(¬**P**) = FNEG (V(**P**))

Ci si può chiedere quale sia la relazione fra un enunciato negato *due volte* e l'enunciato originario. Si consideri ad esempio questo enunciato:

Non si dà il caso che non piova.

Che relazione intercorre fra il valore di verità di questo enunciato e l'enunciato 'Piove'? Se indichiamo con A l'enunciato 'Piove', l'enunciato 'Non si dà il caso che non piova' si può rappresentare simbolicamente così:

¬¬A

Poiché il valore di verità di un enunciato negato una sola volta è correlato al valore di verità dell'enunciato originario tramite FNEG, una negazione doppia, come '¬¬A', deve essere correlata all'originale tramite due applicazioni di FNEG. In altri termini,

V(¬¬A) = FNEG (FNEG(A))

Ne risulta allora che il valore di verità di '¬¬A' coincide con il valore di verità di A. In altre parole, nella logica enunciativa due negazioni formano un'affermazione. In modo informale ciò si può vedere facilmente, poiché ogni applicazione di FNEG inverte il valore di verità. Due applicazioni di FNEG invertono quindi due volte il valore di verità, riportandoci così al valore di verità originario. Ovviamente questo meccanismo può essere iterato:

V(¬**P**)	= FNEG(V(**P**))	
V(¬¬**P**)	= FNEG(FNEG(V(**P**)))	= V(**P**)
V(¬¬¬**P**)	= FNEG(FNEG(FNEG(V(**P**))))	= V(¬**P**)

eccetera.

Si possono fare diverse osservazioni su questo processo. In primo luogo, l'uscita

di una funzione di verità può essere passata in ingresso a un’altra funzione di verità. Questa possibilità permette di concatenare fra loro delle funzioni di verità, come in FNEG(FNEG(FNEG...)).
In secondo luogo, per quanto un enunciato sia complesso, il suo valore di verità è una funzione dei valori di verità degli enunciati che ne costituiscono le *parti verofunzionali*. A volte tale relazione funzionale è espressa da un’unica funzione di verità.
Ad esempio, la relazione che lega V(¬**P**) a V(**P**) è semplicemente:

V(¬**P**) = FNEG(V(**P**))

Altre volte, invece, la relazione funzionale può essere espressa mediante l’uso ripetuto di una funzione di verità. Ad esempio, la relazione che lega V(¬ ¬**P**) a V(**P**) è:

V(¬ ¬**P**) = FNEG(FNEG(V(**P**)))

Il valore di verità di un enunciato negato due volte si ottiene cioè applicando due volte FNEG al valore di verità dell’enunciato **P** stesso. Più in generale, nell’espressione della relazione funzionale fra il valore di verità di un enunciato complesso e i valori di verità delle sue parti, comparirà un’applicazione di una funzione di verità per ogni *connettivo proposizionale* presente nell’enunciato complesso. Oltre alla negazione, in questo capitolo considereremo altri due connettivi proposizionali: la congiunzione e la disgiunzione.

3.2 ENUNCIATI ATOMICI ED ENUNCIATI MOLECOLARI

Alcuni enunciati si presentano in forma molto semplice, essendo costituiti, a volte, soltanto da due parole: il soggetto e il verbo (ad esempio, ‘Fido abbaia’). Ma il semplice conteggio delle parole di un enunciato non sempre può rivelare se l’enunciato è semplice oppure complesso. Un criterio per affrontare questo problema può essere dedotto dall’esame di enunciati negati. Ad esempio, gli enunciati seguenti:

Non si dà il caso che **Q**.
¬**Q**.

sono stringhe di simboli che contengono esplicitamente al proprio interno un enunciato più semplice (in questo caso, **Q**). In prima istanza, possiamo dunque definire *semplice* un enunciato che non contenga enunciati più semplici. Gli enunciati molecolari, al contrario, contengono uno o più enunciati più semplici. Nella

logica enunciativa, considereremo solo enunciati composti costituiti da enunciati collegati fra loro tramite *connettivi verofunzionali*.
Per la rappresentazione degli enunciati più semplici è opportuno riservare le prime lettere dell'alfabeto (eventualmente numerate):

A, B, C,. ..., O, A1, B1, ..., A2, ...

Chiameremo *atomici* questi enunciati. Ogni enunciato costruito a partire da enunciati atomici, per mezzo di connettivi verofunzionali, sarà detto *enunciato molecolare*.
Tenendo presente la distinzione fra enunciati atomici e molecolari, cominciamo ad esaminare i molti modi diversi in cui si possono formare enunciati molecolari a partire da enunciati atomici, oppure da enunciati molecolari più semplici.

3.3 CONGIUNZIONE

Cominciamo con uno dei tipi più semplici e più comuni di enunciati molecolari. L'enunciato

Giorgio è felice e io sono innamorato.

è la *congiunzione* di due enunciati più semplici, ciascuno dei quali è detto *congiunto*. Il primo congiunto è l'enunciato atomico

Giorgio è felice.

e il secondo congiunto è

Io sono innamorato.

In questo esempio entrambi i congiunti sono enunciati atomici; vedremo però nel seguito che ciò non è obbligatorio.
In italiano esistono altri modi per costruire congiunzioni. Anziché usare la parola 'e', avremmo potuto scrivere:

Giorgio è felice ma io sono innamorato.

Anche questo enunciato è una congiunzione. Nella nostra rappresentazione convenzionale delle congiunzioni, esso verrà rappresentato da 'Giorgio è felice e io sono innamorato'. In italiano, questi due enunciati non hanno esattamente lo stesso significato; il 'ma' di solito sta ad indicare che quanto segue è, in qualche modo, il congiunto più importante o rilevante. Ad esempio, posso dire 'Giorgio è felice ma io sono innamorato' se penso che la mia condizione di essere innamorato sia migliore o più importante del fatto che Giorgio è felice.

Per costruire una congiunzione si può usare anche il punto e virgola:

Giorgio è felice; io sono innamorato.

Infine, in italiano alcune altre varianti della congiunzione sono

Io sono innamorato, mentre Giorgio è soltanto felice.

e

Benché Giorgio sia felice, io sono innamorato.

Come già per la negazione, useremo un simbolo speciale anche per la congiunzione: il simbolo usato più spesso è $\wedge$. Anche la scelta delle lettere che rappresentano gli enunciati è estremamente libera, purché si facciano corrispondere lettere diverse ad enunciati diversi.
Una corretta rappresentazione simbolica dell'esempio iniziale 'Giorgio è felice e io sono innamorato' è:

$(A \wedge B)$

ove

A = 'Giorgio è felice.'
B = 'Io sono innamorato.'

e il simbolo $\wedge$ rappresenta la 'e' che li collega. Diremo che $\wedge$ è un *connettivo binario*, perché connette due enunciati per formare un nuovo enunciato molecolare.

LA FUNZIONE DI VERITÀ FCNJ

Come possiamo determinare quando una congiunzione ha valore di verità TRUE? Ad esempio, la congiunzione

Giorgio è felice e io sono innamorato.

quando vale TRUE? Quando vale FALSE? La risposta generale a questa domanda è:

Una congiunzione ha valore di verità TRUE esattamente quando entrambi i congiunti hanno valore di verità TRUE.

Nel caso che stiamo esaminando, 'Giorgio è felice e io sono innamorato' vale

TRUE esattamente quando valgono TRUE sia 'Giorgio è felice', sia 'Io sono innamorato'.
Se entrambi i congiunti hanno valore di verità TRUE, allora la congiunzione ha valore di verità TRUE. Se almeno un congiunto ha valore di verità FALSE, allora la congiunzione ha valore di verità FALSE. Il valore di verità di una congiunzione è funzione dei valori di verità delle sue parti, cioè dei congiunti. Poiché la relazione fra il valore di verità di una congiunzione e i valori di verità delle sue parti è, ancora una volta, di tipo funzionale, possiamo trattare questa nuova funzione di verità in modo analogo alla precedente funzione di verità della negazione.

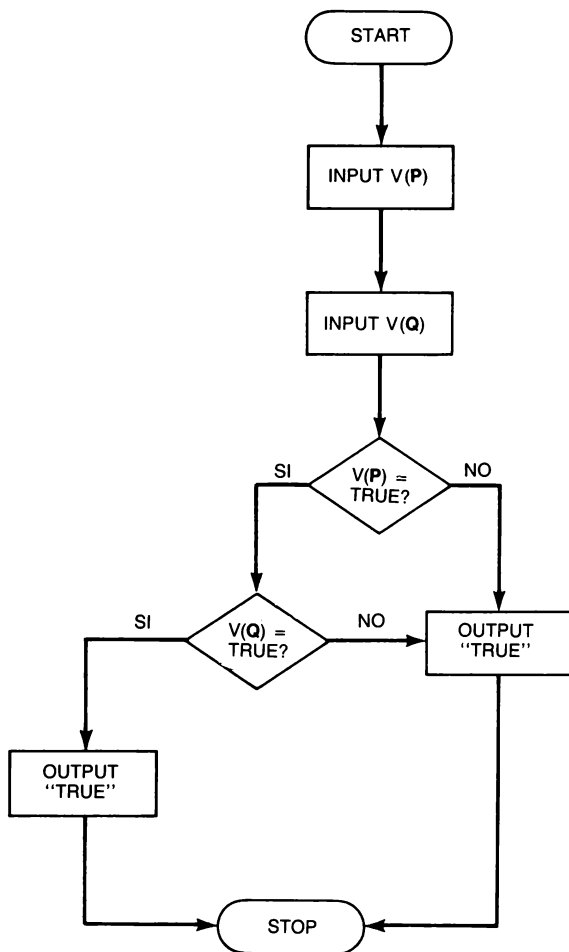



Figura 3.2 Diagramma di flusso per il calcolo di FCNJ

Chiamiamo FCNJ questa funzione. Il valore di verità della congiunzione di due enunciati **P** e **Q** è il risultato dell'applicazione della funzione FCNJ ai valori di verità dei congiunti. Simbolicamente:

V(**P**∧**Q**) = FCNJ(V(**P**), V(**Q**))

Qual è il comportamento di questa funzione di congiunzione? Sappiamo che FCNJ(V(**P**), V(**Q**)) vale TRUE se **P** e **Q** hanno entrambi valore di verità TRUE. Vale invece FALSE se **P** o **Q** ha valore di verità FALSE, oppure se sia **P** che **Q** hanno valore di verità FALSE. In altre parole:

FCNJ(FALSE, FALSE) = FALSE
FCNJ(FALSE, TRUE) = FALSE
FCNJ(TRUE, FALSE) = FALSE
FCNJ(TRUE, TRUE) = TRUE

Questa è una descrizione estensionale completa della funzione di congiunzione. Possiamo rappresentare questa informazione anche sotto forma di tavola di verità:

V(**P**)	V(**Q**)	V(**P**∧**Q**)
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

La funzione di congiunzione FCNJ può anche essere calcolata da un algoritmo, come quello descritto dal diagramma di flusso di Figura 3.2.

Esistono molti altri algoritmi per calcolare FCNJ; ad esempio quello rappresentato nella pagina seguente (Figura 3.3).

Si può anche scrivere un algoritmo per il calcolo di FCNJ con il nostro pseudolinguaggio di programmazione:

```
1. INPUT V(P).
2. INPUT V(Q).
3. IF V(P) = TRUE
     THEN IF V(Q) = TRUE
            THEN GO TO passo 6.
4. OUTPUT "FALSE."
```

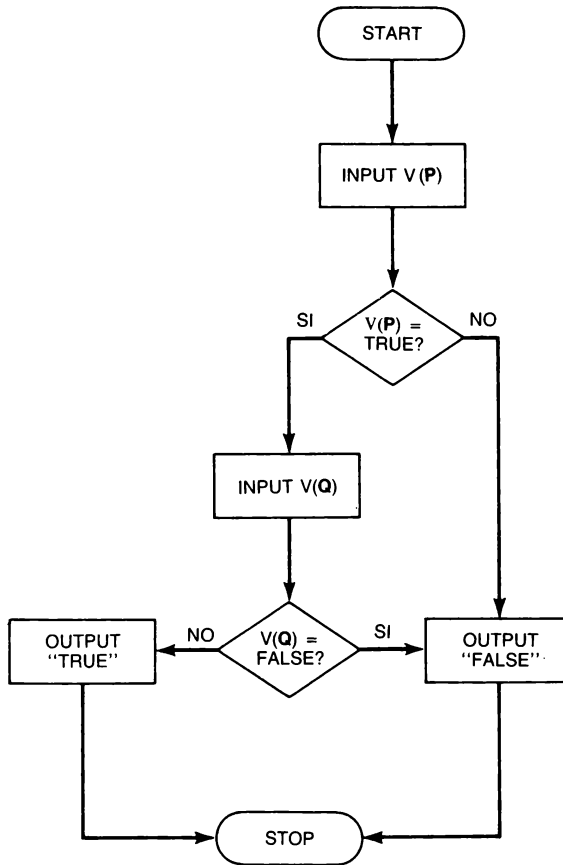



Figura 3.3 Un altro diagramma di flusso per il calcolo di FCNJ

5. STOP.
6. OUTPUT "TRUE."
7. STOP.

Anche con il nostro pseudolinguaggio di programmazione si possono scrivere molti altri algoritmi corretti per il calcolo di FCNJ.
Ad esempio, anche il seguente algoritmo per il calcolo di FCNJ è corretto (si noti che non contiene l'istruzione GO TO):

```
1. INPUT V(P).
2. INPUT V(Q).
3. IF V(P) = TRUE
     THEN IF V(Q) = TRUE
            THEN (a) OUTPUT "TRUE."
                 (b) STOP.
4. OUTPUT "FALSE."
5. STOP.
```

Si può definire un altro algoritmo ricorrendo all'associazione di 1 con TRUE e di 0 con FALSE. Per rendercene conto, riscriviamo dapprima la descrizione estensionale di FCNJ, usando 1 e 0. Otteniamo

FCNJ(0,0) = 0
FCNJ(0,1) = 0
FCNJ(1,0) = 0
FCNJ(1,1) = 1

Dunque, FCNJ vale 1 solo quando entrambi i suoi ingressi sono 1, altrimenti vale 0. Poiché adesso operiamo su numeri, possiamo chiederci quali funzioni *aritmetiche* si comportano in questo modo per i valori 0 e 1.
In effetti, ci sono molte semplici operazioni aritmetiche che si comportano come FCNJ, una volta assegnati i valori dei congiunti in ingresso. Impiegheremo un algoritmo particolare, che può non risultare familiare, ma che è semplice da seguire. Assumiamo come valore di FCNJ(V(**P**), V(**Q**)) il più piccolo fra i due valori V(**P**) e V(**Q**). Ciò corrisponde alla funzione di *minimo*:

MIN(V(**P**), V(**Q**))

Ad esempio,

MIN(2, 5) = 2
MIN(5, 2) = 2
MIN(5, 5) = 5

L'algoritmo è perciò:

1. INPUT V(**P**).
2. INPUT V(**Q**).
3. OUTPUT MIN(V(**P**), V(**Q**)).
4. STOP.

I valori di MIN in corrispondenza degli ingressi che ci interessano sono:

MIN(0, 0) = 0
MIN(0, 1) = 0
MIN(1, 0) = 0
MIN(1, 1) = 1

Poiché la funzione MIN si comporta esattamente come la funzione FCNJ, possiamo ricorrere alla funzione MIN per calcolare la funzione FCNJ.
Quando impieghiamo uno di questi algoritmi, potremmo volerne generalizzare alquanto i primi passi, in modo da poter (1) determinare se l'enunciato composto che stiamo valutando è una congiunzione e (2) scrivere un sottoprogramma che identifichi gli enunciati atomici componenti e ne chieda i valori di verità. Esporremo l'algoritmo completo in un capitolo successivo.
Se abbiamo a che fare con la congiunzione di due enunciati atomici, per trovare il valore di verità della congiunzione possiamo ricorrere direttamente alla definizione di FCNJ. Ma come possiamo trattare una congiunzione di congiunzioni? E che cosa succede se alcuni degli enunciati sono negazioni? Si consideri, ad esempio, una congiunzione della forma

$((\mathbf{P}\wedge\mathbf{Q})\wedge\mathbf{R})$

I suoi due congiunti sono $(\mathbf{P}\wedge\mathbf{Q})$ e **R**. Il primo di questi congiunti, $(\mathbf{P}\wedge\mathbf{Q})$, è a sua volta una congiunzione, mentre il secondo congiunto, **R**, è atomico. Un enunciato con questa struttura logica potrebbe essere:

Sia Giulio che Anna verranno alla festa, così come Luca.

Per determinare il valore di verità di una tale congiunzione, abbiamo ovviamente bisogno di conoscere i valori di verità dei due congiunti $(\mathbf{P}\wedge\mathbf{Q})$ e **R**. Ma per determinare il valore di verità di $(\mathbf{P}\wedge\mathbf{Q})$ dobbiamo prima conoscere i valori di verità sia di **P** che di **Q**. Dunque, per trovare il valore di verità di $((\mathbf{P}\wedge\mathbf{Q})\wedge\mathbf{R})$ dobbiamo trovare prima il valore di verità della congiunzione $(\mathbf{P}\wedge\mathbf{Q})$ e poi il valore di verità della congiunzione di tale enunciato con **R**. La relazione fra il valore di verità di $((\mathbf{P}\wedge\mathbf{Q})\wedge\mathbf{R})$ e i valori di verità di **P**, **Q** e **R**, in base alla descrizione precedente, è:

$$\begin{aligned}
V(\mathbf{P}\wedge\mathbf{Q}) &= FCNJ(V(\mathbf{P}),\ V(\mathbf{Q}))\\
V((\mathbf{P}\wedge\mathbf{Q})\wedge\mathbf{R}) &= FCNJ(V(\mathbf{P}\wedge\mathbf{Q})\qquad\quad,\ V(\mathbf{R}))\\
&= FCNJ(FCNJ(V(\mathbf{P}),\ V(\mathbf{Q})),\ V(\mathbf{R}))
\end{aligned}$$

Si consideri un altro esempio:

$(\neg\mathbf{P}\wedge\mathbf{Q})$

Un enunciato con tale struttura logica potrebbe essere 'Io non sono socialista, ma Mario sì'. Per trovare il valore di verità di questo enunciato (che è ancora una

congiunzione), troviamo dapprima il valore di verità di $\neg$ **P** e poi il valore di verità della congiunzione di esso con il valore di verità di **Q**, cioè con V(**Q**). Ciò significa:

$$V(\neg \mathbf{P} \wedge \mathbf{Q}) = FCNJ(FNEG(V(\mathbf{P})), V(\mathbf{Q}))$$

Si consideri infine questo esempio:

$$\neg(\mathbf{P} \wedge \mathbf{Q})$$

Si tratta della negazione dell'intera congiunzione (**P**$\wedge$**Q**). Un esempio potrebbe essere 'Non si dà il caso che Napoleone fosse sia geniale che pazzo'. Per trovare il valore di verità di un tale enunciato, troviamo dapprima il valore di verità della congiunzione e poi il valore di verità della negazione di esso:

$$V(\neg(\mathbf{P} \wedge \mathbf{Q})) = FNEG(FCNJ(V(\mathbf{P}), V(\mathbf{Q})))$$

I due esempi precedenti mettono in rilievo l'importanza delle parentesi e il fatto che bisogna analizzare con attenzione la struttura logica degli enunciati.
Definiamo con precisione alcuni tipi di stringhe che sono degli enunciati, facendo più attenzione all'uso delle parentesi. Assumiamo qui che gli enunciati non siano frasi in italiano, bensì le loro rappresentazioni mediante lettere singole o particolari combinazioni di lettere singole.

Regole

1. Una stringa costituita da una delle lettere singole (eventualmente numerate) 'A', 'B', ..., 'O', 'A1', 'A2', 'A3', ... è un enunciato.
2. Se una stringa **P** è un enunciato,
   allora la sua negazione $\neg$ **P** è un enunciato.
3. Se le stringhe **P** e **Q** sono enunciati,
   allora la loro congiunzione (**P**$\wedge$**Q**) è un enunciato.

Finora questi sono i soli enunciati che abbiamo a disposizione. Le regole 2 e 3 hanno una precisa corrispondenza in italiano. La regola corrispondente alla 2 afferma che, dato un qualsiasi enunciato in italiano (ad esempio, 'Tutte le mele sono frutti'), anche la sua negazione ('Non si dà il caso che tutte le mele sono frutti') è un enunciato. La regola corrispondente alla 3 afferma che, dati due enunciati qualsiasi in italiano, si può formare un nuovo enunciato che ne è la congiunzione, ad esempio interponendo fra essi la parola 'e'.

3.4 DISGIUNZIONE

In questo capitolo esamineremo un altro connettivo proposizionale binario di uso frequente, correntemente espresso in italiano dalla parola 'o'. Un enunciato costituito da due enunciati collegati da 'o' è una *disgiunzione.* I due enunciati che costituiscono una disgiunzione si chiamano *disgiunti.* Per creare una disgiunzione in italiano, basta prendere due enunciati qualsiasi e scrivere fra essi la parola 'o'. Come nei casi precedenti, useremo un simbolo speciale per rappresentare questo connettivo. Per la disgiunzione, useremo il simbolo '∨'. Dunque, se gli enunciati

1. I tassi di interesse si stabilizzano.
2. Il prezzo dell'oro aumenta.

vengono rappresentati rispettivamente da 'A' e 'B', allora

(A∨B)

significa 'I tassi di interesse si stabilizzano, oppure il prezzo dell'oro aumenta'. Le disgiunzioni hanno valore di verità FALSE esattamente nei casi in cui entrambi i disgiunti hanno valore di verità FALSE. Hanno invece valore di verità TRUE se almeno un disgiunto ha valore di verità TRUE. Le disgiunzioni sono dunque relativamente più "generose" delle congiunzioni, in quanto, a parità di condizioni, hanno valore di verità TRUE in un numero maggiore di casi. Ad esempio, l'enunciato 'I tassi di interesse si stabilizzano, oppure il prezzo dell'oro aumenta' vale FALSE solo quando l'inflazione non si stabilizza e il prezzo dell'oro non aumenta. La disgiunzione ha cioè valore di verità FALSE solo quando entrambi i disgiunti valgono FALSE, altrimenti ha valore di verità TRUE. Usando la nostra notazione,

V(A∨B) = FALSE solo se V(A) = FALSE e V(B) = FALSE

V(A∨B) = TRUE nei casi in cui V(A) o V(B) o entrambi valgono TRUE.

LA FUNZIONE DI VERITÀ FDSJ

Al connettivo proposizionale '∨' è associata un'altra funzione di verità, che chiameremo FDSJ. La funzione FDSJ riceve in ingresso i valori di verità dei due disgiunti e fornisce in uscita il valore di verità dell'intera disgiunzione. La sua descrizione estensionale è:

FDSJ(FALSE, FALSE) = FALSE
FDSJ(FALSE, TRUE) = TRUE
FDSJ(TRUE, FALSE) = TRUE
FDSJ(TRUE, TRUE) = TRUE

Oppure, sotto forma di tavola di verità:

V(**P**)	V(**Q**)	FDSJ(V(**P**), V(**Q**))
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

Ancora una volta, possiamo calcolare questa funzione di verità anche per mezzo di un algoritmo. Nella rappresentazione mediante diagrammi di flusso, un algoritmo che calcola FDSJ è:

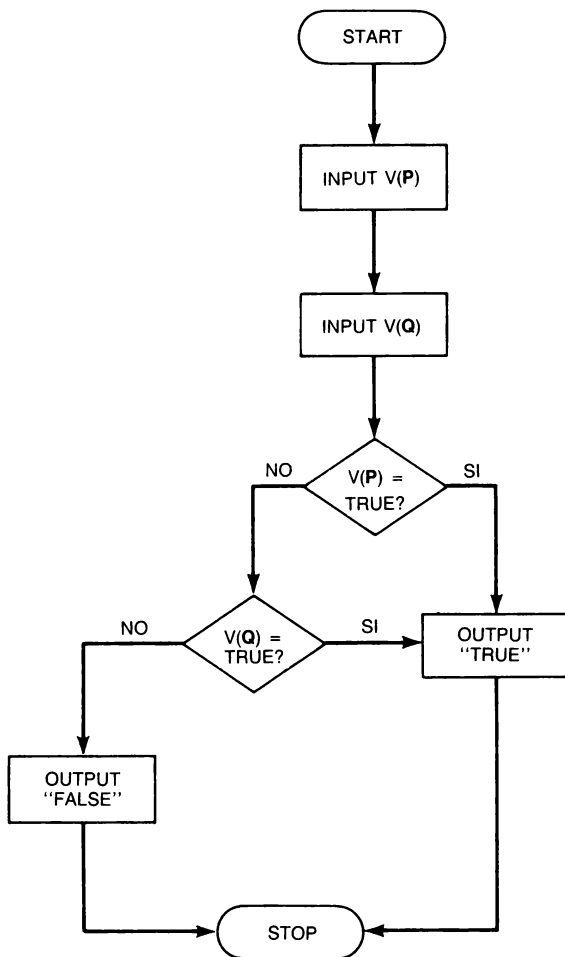



Figura 3.4 Diagramma di flusso per il calcolo di FDSJ

Nel nostro pseudolinguaggio di programmazione, un algoritmo è:

```
1. INPUT V(P).
2. INPUT V(Q).
3. IF V(P) = TRUE
   THEN (a) OUTPUT "TRUE."
        (b) STOP.
4. IF V(Q)= TRUE
   THEN (a) OUTPUT "TRUE."
        (b) STOP.
5. OUTPUT "FALSE."
6. STOP.
```

Considerando le associazioni di FALSE e TRUE con i numeri 0 e 1, la tavola di verità di FDSJ si può scrivere anche:

V(**P**)	V(**Q**)	FDSJ(V(**P**), V(**Q**))
0	0	0
0	1	1
1	0	1
1	1	1

Esistono diverse funzioni aritmetiche che, ricevendo questi ingressi, generano le uscite indicate in questa tavola di verità. Vedremo ora un algoritmo per il calcolo di FDSJ che usa la funzione aritmetica di *massimo*. La funzione di massimo seleziona il più grande dei valori su cui opera. Dunque

MAX(2, 5) = 5
MAX(5, 2) = 5
MAX(5, 5) = 5

L'algoritmo è perciò:

```
1. INPUT V(P).
2. INPUT V(Q).
3. OUTPUT MAX (V(P), V(Q)).
4. STOP.
```

Se applicato ai valori di verità 0 e 1, questo algoritmo fa esattamente ciò che fa FDSJ.

DISGIUNZIONE INCLUSIVA E DISGIUNZIONE ESCLUSIVA

Spesso una disgiunzione in italiano deve essere riscritta, affinché si possa evidenziare il fatto che si tratta della disgiunzione di due enunciati. Si consideri la frase:

Il vincitore è Mario oppure Andrea.

Occorre riscriverla nella forma:

Il vincitore è Mario oppure il vincitore è Andrea.

per vedere che si tratta della disgiunzione di due enunciati.
Un enunciato come quello sopra riportato, tuttavia, viene spesso inteso in senso *esclusivo*, ossia si considera che esso esprima il fatto che il vincitore è o Mario o Andrea, *ma non entrambi*. Finora, trattando le disgiunzioni abbiamo considerato solo il senso *inclusivo*, ossia quello che include la possibilità che entrambi i disgiunti valgano TRUE. Ad esempio, questo uso della disgiunzione per l’enunciato sopra riportato ammette la possibilità che abbiano vinto sia Mario che Andrea. Nel capitolo seguente considereremo un connettivo apposito per la disgiunzione intesa in senso esclusivo.

3.5 ENUNCIATI CON PIÙ CONNETTIVI

A questo punto, è opportuno affrontare alcuni esempi per capire bene come si trattano gli enunciati che contengono insieme negazioni, congiunzioni e disgiunzioni. Prima di considerare gli esempi, però, dobbiamo aggiungere un’altra regola a quelle che servono per determinare quali stringhe sono enunciati:

4. Se le stringhe **P** e **Q** sono enunciati,
   allora la loro disgiunzione **(P∨Q)** è un enunciato.

In base a questa nuova regola (e alle regole 2 e 1 del paragrafo 3.3),

(¬A∨B)

è un enunciato, perché:

‘A’ e ‘B’ sono entrambi enunciati (per la regola 1).
Dunque, ‘¬A’ è un enunciato (per la regola 2).
Perciò, ‘(¬A∨B)’ è un enunciato (per la regola 4).

Il suo valore di verità si può calcolare con questa espressione

FDSJ(FNEG(V(A)), V(B))

determinata in base al seguente ragionamento: ad ogni connettivo proposizionale è univocamente associata una funzione di verità (FNEG per $\neg$, FCNJ per $\wedge$, FDSJ per $\vee$). Operando "dall'interno verso l'esterno" sull'enunciato originario, troviamo dapprima il valore di verità di '$\neg$A':

V($\neg$A) = FNEG(V(A))

Si vede poi che

V($\neg$A$\vee$B) = FDSJ(V($\neg$A), V(B))

Sostituendo V($\neg$A) con la sua espressione precedentemente trovata si ottiene

V($\neg$A$\vee$B) = FDSJ(FNEG(V(A)), V(B))

Si consideri l'enunciato ((A $\vee$ B) $\wedge\neg$C) (si verifichi che si tratta effettivamente di un enunciato in base alle regole dalla 1 alla 4). Un'analisi passo per passo di questo enunciato conduce all'espressione:

V((A$\vee$B) $\wedge\neg$C) = FCNJ(FDSJ(V(A), V(B)), FNEG(V(C)))

Siamo giunti a questa formula ragionando come nell'esempio precedente, associando ad ogni connettivo proposizionale la funzione di verità appropriata. Si valuti questa espressione per diversi valori di verità di A, B e C.
Ad esempio, quando V(A) = TRUE, V(B) = FALSE e V(C) = TRUE, il suo valore è FALSE. Siamo giunti a questa risposta nel modo seguente. Quando V(A) = TRUE e V(B) = FALSE, l'uscita della funzione FDSJ(V(A), V(B)) è TRUE. Perciò, sostituendo 'FDSJ(V(A), V(B))' con 'TRUE' si ha

FCNJ(TRUE, FNEG(V(C)))

Quando V(C) = TRUE, FNEG(V(C)) vale FALSE. Dunque, sostituendo 'FNEG(V(C))' con 'FALSE' si ha:

FCNJ(TRUE, FALSE)

Se gli ingressi sono TRUE e FALSE, l'uscita della funzione FCNJ è FALSE.
Quando invece V(A) = TRUE, V(B) = FALSE e V(C) = FALSE, il valore di verità dell'enunciato è TRUE.
Un altro esempio è:

(A $\wedge\neg$(B$\vee$C))

Il valore di verità di (B$\vee$C) è FDSJ(V(B), V(C)), e il valore di verità di $\neg$(B$\vee$C) è:

FNEG(FDSJ(V(B), V(C)))

Perciò l'intero enunciato ha valore di verità

FCNJ(V(A), FNEG(FDSJ(V(B), V(C))))

3.6 ALTRI CONNETTIVI

I tre connettivi che abbiamo esaminato non sono gli unici possibili. Esistono molti altri modi per formare enunciati nuovi a partire da enunciati vecchi. Alcuni di questi modi usano connettivi verofunzionali, altri no. Un esempio di connettivo *non* verofunzionale è costituito dal connettivo unario 'Il mio amico crede che ...'. Può sembrare un connettivo strano, ma si comporta in modo simile al connettivo unario verofunzionale 'Non si dà il caso che'. Per capire perché 'Il mio amico crede che' è verofunzionale, si consideri l'enunciato A = 'Il mio amico crede che la nostra squadra vincerà la partita'. Il suo valore di verità non è una funzione del valore di verità di B = 'La nostra squadra vincerà la partita', perché V(A) può essere TRUE sia se V(B) = TRUE, sia se V(B) = FALSE. Il valore di verità di A non dipende cioè soltanto dal valore di verità di B. Più precisamente, non esiste nessun algoritmo che fornisca V(A) in uscita ricevendo solo V(B) in ingresso.
Comunque, oltre ai connettivi discussi in questo capitolo, ne esistono molti altri che possono essere associati a una funzione di verità. I più importanti fra questi altri connettivi sono 'se ... allora' e, soprattutto per l'informatica, 'NAND' e 'NOR'. Come 'non', 'e' ed 'o', ciascuno di questi connettivi può essere espresso in italiano con molte parole diverse. Non abbiamo cercato di rendere automatica la traduzione dall'italiano alla notazione simbolica o viceversa. Forse non sarebbe possibile. Questo è un ambito che, per ora, sembra debba essere riservato a processi non automatici (per un tentativo di renderlo automatico vedi però Otto, 1978; per una discussione della connessa ricerca sulla "linguistica computazionale" — una branca dell'informatica che si occupa della comprensione del linguaggio naturale — vedi Winograd, 1983).
Esiste tuttavia un aspetto in base al quale la discussione sui tre connettivi introdotti in questo capitolo può considerarsi completa. Si può cioè dimostrare che ogni proposizione che si possa voler esprimere può essere espressa da un enunciato che usa solo negazione, congiunzione e disgiunzione. Anzi, in realtà sono sufficienti la negazione e uno solo degli altri due connettivi. Un'ulteriore discussione su questo punto potrà però avere luogo soltanto in un capitolo successivo.

3.7 CONCLUSIONI

In questo capitolo abbiamo cominciato a studiare la *logica enunciativa*. Abbiamo considerato un enunciato come una stringa di simboli che esprime una proposizione. Abbiamo visto come si costruiscono enunciati *molecolari* complessi a partire da enunciati molecolari più semplici, oppure a partire da enunciati *atomici*, per mezzo di tre connettivi: *negazione, congiunzione* e *disgiunzione*.
Esistono molti metodi per negare un enunciato espresso in italiano; un metodo convenzionale consiste nel prefiggere l'espressione 'Non si dà il caso che' all'enunciato stesso. Se **P**, nella nostra notazione simbolica, è un enunciato, allora ¬**P** è la sua negazione. Il valore di verità di una negazione è TRUE (FALSE) se il valore di verità dell'enunciato non negato è FALSE (TRUE).
Una *funzione* associa un'unica uscita ad ogni insieme assegnato di ingressi. La negazione è un connettivo *verofunzionale*; FNEG è la funzione che in uscita fornisce TRUE (FALSE) quando il suo ingresso è FALSE (TRUE). Le funzioni possono essere descritte *estensionalmente*, elencando tutti i possibili ingressi e le corrispondenti uscite. Nel caso di una *funzione di verità*, come FNEG, gli ingressi e le uscite possono essere rappresentati in una *tavola di verità*. Le funzioni possono anche essere descritte *intensionalmente*, fornendo un algoritmo per calcolare l'uscita che corrisponde a ciascun insieme di ingressi. Nel caso di FNEG, abbiamo considerato algoritmi che operano in modo aritmetico, sottraendo da 1 l'ingresso (assunto pari a 0 se FALSE, a 1 se TRUE).
Il nostro metodo convenzionale per formare una congiunzione in italiano consiste nel collegare due enunciati mediante la parola 'e'; in forma simbolica, se **P** e **Q** sono enunciati, allora (**P**∧**Q**) è un enunciato. Il valore di verità della congiunzione di due enunciati è TRUE se i valori di verità di entrambi i *congiunti* sono TRUE; altrimenti è FALSE. FCNJ è la funzione di verità che in uscita fornisce TRUE quando i suoi ingressi sono entrambi TRUE; altrimenti restituisce FALSE. Abbiamo considerato algoritmi per il calcolo di FCNJ che usano la funzione aritmetica MIN.
Se **P** e **Q** sono enunciati, allora la loro disgiunzione è l'enunciato (**P**∨**Q**) (in italiano si userebbe la parola 'o'). Il valore di verità della disgiunzione di due enunciati è TRUE se almeno uno dei due *disgiunti* ha valore di verità TRUE; altrimenti è FALSE. In questo modo, la disgiunzione è intesa in senso *inclusivo*, poiché ammette la possibilità che entrambi i disgiunti abbiano valore di verità TRUE. La funzione di verità FDSJ in uscita fornisce FALSE quando entrambi i suoi ingressi sono FALSE; altrimenti restituisce TRUE. Gli algoritmi considerati per descrivere FDSJ intensionalmente usano la funzione aritmetica MAX.
Al termine abbiamo fornito quattro regole per determinare se una stringa è un enunciato, e abbiamo trattato il problema della determinazione del valore di verità di un enunciato molecolare contenente più di un connettivo.

3.8 ESERCIZI

A. Tenendo conto delle abbreviazioni seguenti, esprimere in italiano, nella forma più chiara possibile, il significato delle formule riportate.
A = ‘Piove.’
B = ‘Il cielo è sereno.’
C = ‘Nevica.’
D = ‘Fa freddo.’
Esempio: $\neg(A \wedge B)$
Risposta: Non si dà il caso che piova e, allo stesso tempo, il cielo sia sereno.
1. $\neg C$
2. $\neg\neg D$
3. $(A \wedge C)$
4. $(B \vee D)$
5. $(\neg B \wedge D)$
6. $(B \wedge \neg D)$
7. $\neg(A \vee C)$
8. $\neg(B \wedge C)$
9. $(A \wedge (B \vee C))$
10. $(\neg B \vee (C \wedge D))$
11. $(C \wedge (\neg D \vee A))$
12. $((A \vee B) \wedge (C \vee D))$
13. $\neg(B \vee D)$
14. $(\neg B \wedge \neg D)$

B. Definire il valore di verità di ciascuno degli enunciati dell’esercizio (A), assumendo che:

V(‘Piove.’) = V(A) = TRUE
V(‘Il cielo è sereno.’) = V(B) = FALSE
V(‘Nevica.’) = V(C) = TRUE
V(‘Fa freddo.’) = V(D) = FALSE

Esempio: $\neg(A \wedge B)$
Risposta:TRUE [Il valore di verità di $(A \wedge B)$ è FALSE perché B vale FALSE. Dunque il valore di verità di $\neg(A \wedge B)$ è il valore di verità opposto, ossia è TRUE.]

C. Rappresentare simbolicamente gli enunciati sotto riportati, usando le seguenti abbreviazioni:
A = ‘I calcolatori pensano.’
B = ‘I calcolatori sono dotati di sensibilità.’
C = ‘Gli animali pensano.’
D = ‘Gli animali sono dotati di sensibilità.’
Esempio: I calcolatori non pensano.
Risposta: $\neg A$
1. I calcolatori pensano e sono dotati di sensibilità.
2. Gli animali sono dotati di sensibilità, ma i calcolatori no.

3. O i calcolatori o gli animali pensano.
4. Non si dà il caso che gli animali non siano dotati di sensibilità.
5. O i calcolatori o gli animali pensano e sono anche dotati di sensibilità.
6. O gli animali pensano, o i calcolatori non pensano ma sono dotati di sensibilità.
7. Né gli animali né i calcolatori pensano.
8. O gli animali sono dotati di sensibilità, o i calcolatori non lo sono.
9. I calcolatori e gli animali pensano e sono dotati di sensibilità.
10. O i calcolatori e gli animali pensano, oppure i calcolatori o gli animali sono dotati di sensibilità.
11. Né i calcolatori né gli animali pensano o sono dotati di sensibilità.
12. Né i calcolatori né gli animali pensano e sono anche dotati di sensibilità.

D. Determinare il valore di verità di ciascuno degli enunciati sopra riportati, assumendo che:
V('I calcolatori pensano.') = TRUE
V('I calcolatori sono dotati di sensibilità.') = FALSE
V('Gli animali pensano.') = FALSE
V('Gli animali sono dotati di sensibilità.') = TRUE
Esempio: I calcolatori non pensano.
$\neg A$
Risposta: FALSE, perché il valore di verità di $\neg A$ è l'opposto del valore di verità di A.

E. A volte i valori di verità degli enunciati atomici costituenti un enunciato molecolare non sono prefissati. È però possibile definire un algoritmo che determini il valore di verità di un enunciato molecolare a partire da ingressi costituiti dai valori di verità degli enunciati atomici.
Esempio: $\neg(A \wedge B)$
Risposta: Un algoritmo che usa funzioni aritmetiche è rappresentato mediante un diagramma di flusso nella Figura 3.5.

Un algoritmo che non usa funzioni aritmetiche può essere, nel nostro pseudolinguaggio di programmazione:

```
1. INPUT V(A).
2. IF V(A) = FALSE
     THEN (a) OUTPUT "TRUE."
          (b) STOP.
3. INPUT V(B).
4. IF V(B) = FALSE
     THEN (a) OUTPUT "TRUE."
          (b) STOP.
5. OUTPUT "FALSE."
6. STOP.
```

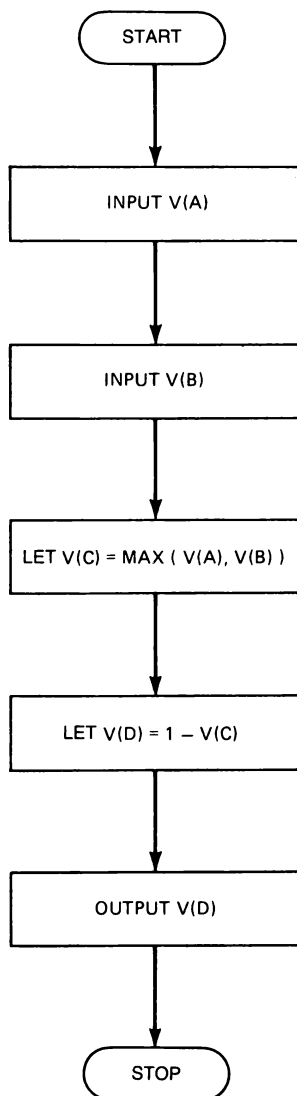



Figura 3.5 Diagramma di flusso per l'Esercizio E

Costruire un algoritmo per ciascuna delle formule seguenti, scegliendo se usare i diagrammi di flusso o lo pseudolinguaggio di programmazione.

1. $\neg\neg A$
2. $(A \wedge \neg B)$
3. $(\neg A \vee B)$
4. $((A \wedge B) \vee C)$
5. $(\neg A \wedge (B \vee C))$
6. $((A \wedge \neg B) \vee (\neg C \vee D))$
7. $\neg(A \vee B)$
8. $(\neg A \vee A)$
9. $(B \wedge \neg B)$

F. Determinare le funzioni di verità che calcolano i valori di verità degli enunciati molecolari sotto riportati.
Esempio: $(A \vee \neg B)$
Risposta: FDSJ(V(A), FNEG(V(B)))
Spiegazione: Si opera dall'interno verso l'esterno, cioè si comincia dalla sottoformula più interna. In questo caso, l'enunciato più interno è '$\neg$B'.
Il valore di verità di '$\neg$B' è dato da:
FNEG(V(B))
Ma '$\neg$B' fa parte di una disgiunzione, il cui valore di verità è calcolato da:
FDSJ(V(A), V($\neg$B))
Poiché V($\neg$B) = FNEG(V(B)), per sostituzione si ha:
FDSJ(V(A), FNEG(V(B)))

1. $\neg(A \vee B)$
2. $\neg(\neg A \vee B)$
3. $(A \wedge B)$
4. $(\neg A \wedge B)$
5. $\neg(A \wedge B)$
6. $\neg(\neg A \wedge \neg B)$
7. $(A \wedge (B \vee C))$
8. $(\neg B \vee (C \wedge \neg D))$
9. $(\neg B \wedge (\neg C \wedge \neg D))$
10. $\neg(A \wedge (\neg B \vee C))$

G. Assegnati i seguenti valori di verità:
V(A) = 1
V(B) = 0
V(C) = 0
V(D) = 1
calcolare l'uscita di ciascuna delle funzioni sotto elencate.
Esempio: MAX(V(A), 1 − (1 − V(B)))
Risposta: 1 (cioè TRUE)

Spiegazione: Il valore di verità di A è 1. Sostituendo dunque V(A) con 1, l’espressione diventa:

MAX(1, 1 − (1 − V(B)))

Poiché il secondo ingresso di MAX può valere solo o 0 o 1, sappiamo già che il massimo è 1, senza aver bisogno di proseguire. Se proseguissimo, sostituiremmo V(B) con 0, ottenendo:

MAX(1, 1 − (1 − 0))

ossia

MAX(1, 1 − 1)

vale a dire

MAX(1, 0)

Ma il massimo fra 1 e 0 è appunto

1

proprio come avevamo precedentemente concluso.

1. 1 − (1 − V(B))
2. 1 − MAX(V(A), V(B))
3. FCNJ(V(A), V(B))
4. MIN(1 − V(A), V(B))
5. FCNJ(1 − V(B), 1 − V(C))
6. MAX(V(B), 1 − V(D))
7. FCNJ(V(A), FDSJ(V(B), V(C)))
8. MAX(V(B), MAX(V(C), V(D)))
9. FCNJ(1 − FDSJ(V(A), V(B)), V(A))
10. MIN(1 − MAX(V(B), V(C)), MIN(V(A), 1 − V(D)))
11. 1 − FDSJ(1 − V(B), FCNJ(FDSJ(V(B), V(C)), 1 − V(D)))
12. 1 − (1 − (1 − (FCNJ(V(A), V(B))))

H. Per ogni espressione dell’esercizio precedente, determinare l’enunciato corrispondente.

Esempio: 1 − MIN(V(A), V(B))

Risposta: $\neg(A \wedge B)$

Spiegazione: Si opera dall’esterno verso l’interno. Un’espressione che comincia con ‘1 − ’ corrisponde certamente a una negazione, quindi possiamo scrivere un segno di negazione:

$\neg$

La successiva funzione che incontriamo è

MIN(V(A), V(B))

L’enunciato corrispondente è

$(A \wedge B)$

Aggiungendolo al precedente segno di negazione otteniamo

$\neg(A \wedge B)$

I. Negazione.

Scrivere un algoritmo per il calcolo del valore di verità di una negazione, usando una funzione aritmetica diversa dalla funzione 1 − V(**P**) usata nel testo.

J. Congiunzione.
  1. Scrivere due algoritmi per il calcolo del valore di verità di una congiunzione, usando funzioni aritmetiche diverse da quelle usate nel testo.
  2. Riscrivere uno degli algoritmi per il calcolo di FCNJ del paragrafo 3.3, mettendo V(**P**) = FALSE come primo test.

K. Disgiunzione.
  1. Scrivere un algoritmo per il calcolo del valore di verità di una disgiunzione, usando una funzione aritmetica diversa da quella usata nel testo.
  2. Riscrivere l'algoritmo per il calcolo di FDSJ del paragrafo 3.4, mettendo V(**P**) = FALSE come primo test.

L. Abbiamo definito i concetti di congiunzione e di disgiunzione solo per due enunciati. Analogamente, abbiamo considerato 'e' ed 'o' come collegamenti fra due soli enunciati espressi in italiano. A volte potrebbe invece essere utile considerare una funzione di verità o un connettivo più generale, che possa ricevere in ingresso due o più valori di verità, oppure collegare due o più enunciati. Si consideri questo enunciato:

   Roberto è trentenne, biondo e sposato.

   Quando ha valore di verità TRUE? Quale funzione aritmetica (diversa da quella usata nel testo) si potrebbe usare in un algoritmo per calcolare il valore di verità di un enunciato con più di due congiunti?

M. 1. Usando solo congiunzione e negazione, costruire un enunciato che abbia valore di verità FALSE solo quando A, B e C hanno tutti valore di verità TRUE.
  2. Usando solo negazione e congiunzione, scrivere un enunciato che valga TRUE quando uno solo fra A e B vale TRUE, e che valga FALSE negli altri casi.
  3. Usando solo negazione e congiunzione, scrivere un enunciato che valga FALSE solo quando A e B valgono o entrambi TRUE, o entrambi FALSE.
  4. Usando solo congiunzione e negazione, scrivere un enunciato che valga TRUE quando, fra A, B e C, esattamente due valgono FALSE, e che valga FALSE negli altri casi.

3.9 SUGGERIMENTI PER UNA REALIZZAZIONE SU CALCOLATORE

Benché l'ideazione di algoritmi per il calcolo di FNEG, FCNJ e FDSJ non sia particolarmente complicata né, a questo punto, particolarmente utile, questi algoritmi risulteranno più avanti di grande importanza. Senza di essi, infatti, non saremmo in grado di determinare il valore di verità di un enunciato molecolare arbitrario.

A seconda del calcolatore e del linguaggio di programmazione disponibile, esistono diversi modi per progettare un programma che calcoli queste funzioni di verità. Per esempio, in Pascal e nei dialetti del BASIC in cui le definizioni di funzioni possono estendersi su più righe ("definizione di funzione multi-istruzione"), il problema è di semplice soluzione: FNEG, FCNJ e FDSJ possono essere definite quasi esattamente come indicato negli algoritmi del testo scritti nel nostro pseudo-linguaggio di programmazione.
Se le definizioni di funzioni su più righe non sono possibili, occorre una strategia più attenta. Forse il metodo più semplice consiste nel rifarsi all'associazione di 1 con TRUE e di 0 con FALSE. Tale convenzione di solito permette di far stare su una sola riga la definizione di una funzione di verità.
A volte si possono usare direttamente le operazioni booleane predefinite NOT ("non"), AND ("e") e OR ("o"). Altrimenti ci si può rifare alle funzioni aritmetiche menzionate nel testo. Siano p e q i valori aritmetici di verità di **P** e **Q** rispettivamente. Allora:

Come FNEG(p) si assuma $1 - p$.
Come FCNJ(p, q) si assuma MIN(p, q).
Come FDSJ(p, q) si assuma MAX(p, q).

Può accadere che le funzioni MIN e MAX debbano a loro volta essere definite, oppure che convenga usare altre funzioni aritmetiche che abbiano lo stesso comportamento di FCNJ e FDSJ; ad esempio:

Come FCNJ(p, q) si assuma $(p*q)$.
Come FDSJ(p, q) si assuma $(p+q) - (p*q)$.

ove, come spesso in informatica, * è l'operatore di moltiplicazione. La coincidenza della funzione di congiunzione con la moltiplicazione e della funzione di disgiunzione con un tipo speciale di somma fu scoperta da George Boole fra il 1830 e il 1850, e costituisce il fondamento di un'area della matematica detta appunto algebra di Boole.
Anziché una definizione di funzione, si può usare qualche tipo di sottoprogramma esplicito, come il GOSUB in BASIC o le procedure in Pascal. Ogni volta che vogliamo calcolare una funzione di verità, passiamo il controllo a un altro punto del programma, che esegue le operazioni necessarie e restituisce poi il controllo al punto originario del programma. Ad esempio, in un BASIC limitato per microcalcolatori, per calcolare il valore di verità di una congiunzione (A$ ∧ B$), ove A$ e B$ sono i congiunti e A e B sono i rispettivi valori di verità, si potrebbe avere:

```
100 INPUT A
200 INPUT B          [Riceve in ingresso e memorizza nelle
                     variabili A e B i valori di verità di A$
                     e B$]
300 LET P = A
```

```
 400 LET Q = B            [Passa i valori di A e B alle variabili
                          P e Q usate nel sottoprogramma]
 500 GOSUB 1000           [Passa il controllo alla riga 1000 se in-
                          contriamo una congiunzione]
1000 IF P = 0 THEN GOTO 1400
1100 IF Q = 0 THEN GOTO 1400
1200 LET R = 1            [La congiunzione ha valore di verità
                          TRUE, cioè 1]
1300 RETURN
1400 LET R = 0            [La congiunzione ha valore di verità
                          FALSE, cioè 0]
1500 RETURN
```

ove l'istruzione 'RETURN' restituisce il controllo all'istruzione successiva al GOSUB (qui non rappresentata).
Il valore della congiunzione quando il sottoprogramma restituisce il controllo viene memorizzato nella variabile di nome R. In un vero programma, la riga 500 sarebbe in realtà più complessa, perché dovrebbe utilizzare il sottoprogramma che inizia alla riga 1000 solo dopo aver verificato di aver a che fare con una congiunzione. Le variabili P e Q sono necessarie se si vuole applicare il sottoprogramma a due congiunti qualsiasi (A$ e B$, C$ e D$, eccetera) in qualunque punto del programma. In tal modo, l'informazione sui valori di verità dei due congiunti, indipendentemente dai loro nomi originari, viene temporaneamente memorizzata in P e Q.

4

Logica enunciativa: il connettivo 'se ... allora ...' e altri connettivi supplementari

Nel capitolo precedente abbiamo considerato il connettivo unario 'non' ($\neg$) e i connettivi binari 'e' ($\wedge$) ed 'o' ($\vee$). Passiamo ora ad esaminare diversi altri connettivi binari: 'se ... allora ...' ($\rightarrow$), 'se e solo se' ($\leftrightarrow$), disgiunzione esclusiva (XOR), 'né ... né ...' (NOR) e 'non insieme ... e ...' (NAND). Forniremo infine un elenco completo di tutti i possibili connettivi binari.
A rigore, questi connettivi supplementari non sono necessari. I due connettivi $\neg$ e $\wedge$ sono infatti sufficienti per esprimere qualunque enunciato in logica enunciativa. Tuttavia, questi connettivi supplementari che vedremo corrispondono ad espressioni di uso corrente in italiano; perciò l'aggiunta di appositi simboli per questi connettivi permette di tradurre direttamente in forma simbolica molti enunciati espressi in italiano. Quando però dobbiamo decidere se introdurre tali connettivi supplementari, ci troviamo di fronte a un compromesso. Se usiamo pochissimi connettivi, otteniamo una rappresentazione logica compatta, elegante e facile da capire per un calcolatore, ma non sempre adeguata per un utente umano, abituato a pensare in un linguaggio naturale come l'italiano. Viceversa, se introduciamo una gamma completa di simboli corrispondenti a connettivi di uso corrente in italiano, otteniamo una rappresentazione logica che risulta familiare agli esseri umani; tale rappresentazione è però ripetitiva, certe espressioni verofunzionali sinonimiche sono difficili da identificare, e occorre un lavoro di programmazione più gravoso per rendere comprensibile a un calcolatore tale notazione. Nei capitoli successivi incontreremo altre decisioni simili, che richiedono un compromesso fra le esigenze degli esseri umani e quelle dei calcolatori, nonché fra i vantaggi di tipi di algoritmi diversi.

4.1 L'ENUNCIATO CONDIZIONALE

Nella conversazione corrente, a volte limitiamo un'affermazione usando una frase che comincia con 'se'. Tali affermazioni sono dette *condizionali*. Anziché promettere direttamente un gelato, potrei dire:

Se vieni con me alla partita, ti offro un gelato.

L'aspetto importante di questo esempio consiste nel fatto che io non mantengo la promessa *soltanto* nel caso in cui tu venga alla partita con me, ma io non ti offra un gelato. Supponiamo invece che tu non venga alla partita con me. In tal caso, io non posso mancare alla mia promessa, né se ti offro lo stesso il gelato né se non te lo offro, perché la mia promessa era condizionale. Poiché non si è verificata la condizione che tu venissi alla partita con me, non ho alcun modo di mancare alla mia promessa.
Gli enunciati che riguardano eventi futuri sono tipicamente espressi in forma condizionale:

Se la Roma perde la prossima partita, allora la Juventus vince lo scudetto.

Quando ha valore di verità FALSE questo enunciato? Ovviamente ha valore di verità FALSE nel caso in cui la Roma perda la prossima partita, ma la Juventus non vinca lo scudetto. E se la Roma vince la prossima partita? Il condizionale ha valore di verità TRUE o FALSE? Oppure abbiamo trovato un tipo di enunciato il cui valore di verità non è né TRUE né FALSE?
Se la Roma vince la prossima partita, esistono parecchi motivi per assumere che il valore di verità dell'intero enunciato condizionale sia TRUE. Un motivo consiste nell'analogia con la promessa condizionale, che, come abbiamo visto, si considera rotta solo quando la condizione ha valore di verità TRUE e l'azione promessa non viene compiuta.
Per chiarire le cose, ribadendo che il valore di verità di tutti gli enunciati che consideriamo dipende solo dai valori di verità dei loro componenti, assumeremo che tutti gli enunciati della forma

Se **P**, allora **Q**.

abbiano valore di verità FALSE in tutti i casi in cui **P** vale TRUE e, allo stesso tempo, **Q** vale FALSE. In ogni altro caso, l'enunciato condizionale ha valore di verità TRUE. Un condizionale che goda di tale proprietà si dice *condizionale materiale*. Un enunciato condizionale è costituito da due parti, ciascuna delle quali è, a sua volta, un enunciato (proprio come avviene nel caso delle congiunzioni e delle disgiunzioni, ma non nel caso delle negazioni, che hanno un solo enunciato componente). L'enunciato che definisce la condizione, solitamente posto subito dopo la parola 'se', è detto *antecedente* del condizionale. La seconda parte del

condizionale, che spesso segue la parola 'allora', è detta *conseguente* del condizionale.
In logica interessano soprattutto i condizionali i cui conseguenti sono enunciati dichiarativi, piuttosto che i condizionali i cui conseguenti sono azioni promesse, minacce o ordini. In informatica, viceversa, il condizionale più frequente è quello il cui conseguente è un'istruzione: se <condizione> allora <istruzione>.
Abbiamo già visto un condizionale il cui conseguente è una promessa. Altre varianti sono costituite da condizionali aventi come conseguenti delle minacce oppure degli ordini:

> Se mi prendi il libro, lo dico alla maestra.
> Se non puoi fare a meno di ridere, esci.

Oltre ai condizionali materiali, che sono interamente composti da enunciati dichiarativi, esistono dunque almeno altri tre tipi di condizionali: promesse condizionali, minacce condizionali e ordini condizionali. Un esame di questi ultimi può aiutare a capire meglio quando un enunciato condizionale materiale ha valore di verità TRUE oppure FALSE. Si considerino le seguenti affermazioni sulle promesse, le minacce e gli ordini:

> Una promessa condizionale non viene mantenuta esclusivamente nei casi in cui l'antecedente ha valore di verità TRUE, ma l'azione descritta nel conseguente non viene compiuta; altrimenti la promessa è mantenuta.
> A una minaccia condizionale non viene dato seguito esclusivamente nei casi in cui l'antecedente ha valore di verità TRUE, ma l'azione descritta nel conseguente non viene compiuta; altrimenti la minaccia rimane valida.
> Un ordine condizionale viene disatteso esclusivamente nei casi in cui l'antecedente ha valore di verità TRUE, ma l'azione descritta nel conseguente non viene compiuta; altrimenti l'ordine non è disatteso.

Anche il valore di verità di un enunciato condizionale materiale segue esattamente questo modello:

> Un enunciato condizionale materiale ha valore di verità FALSE esclusivamente quando l'antecedente vale TRUE ma il conseguente vale FALSE; altrimenti il condizionale materiale non ha valore di verità FALSE.

Gli enunciati, però, possono avere solo i valori di verità TRUE e FALSE. Dunque, se un enunciato condizionale materiale non ha valore di verità FALSE, esso ha valore di verità TRUE.
In italiano, oltre al condizionale materiale, esistono diversi altri tipi di enunciati condizionali. Gli enunciati condizionali possono essere usati per esprimere nessi causali fra due eventi, come 'Se la temperatura dell'acqua raggiunge i 100°C, allo-

ra l'acqua bolle'. Possono essere anche usati per esprimere situazioni non corrispondenti ai dati di fatto: 'Se il fiammifero non fosse stato bagnato, si sarebbe acceso quando l'hai sfregato'. A volte sono poi usati per esprimere connesioni temporali: 'Se cinque minuti fa erano le tre, adesso sono le tre e cinque'.
Nei condizionali materiali non si considera che esistano tali connessioni fra antecedente e conseguente. Un buon esempio di condizionale materiale "puro" potrebbe essere 'Se la nostra squadra vince, io sono la regina d'Inghilterra'. D'ora in poi, gli unici enunciati condizionali che studieremo saranno quelli materiali, anche quando abbiano l'aspetto dei condizionali di uno degli altri tipi.
Forse la cosa più importante da tener presente quando si valuta un condizionale è il fatto che non bisogna confondere il valore di verità dell'antecedente con il valore di verità dell'intero condizionale. In altre parole, bisogna fare una distinzione fra enunciati come

I cavalli hanno le ali.

e

Se i cavalli hanno le ali, allora volano.

Il primo enunciato, che è atomico, ha valore di verità FALSE: i cavalli non hanno le ali. Ciò non significa però che il condizionale ('Se i cavalli hanno le ali, allora volano') abbia valore di verità FALSE. Il condizionale ha invece valore di verità TRUE appunto perché il suo antecedente ha valore di verità FALSE. Il valore di verità di un condizionale materiale dipende solo dai valori di verità del suo antecedente e del suo conseguente, e non da una connessione causale, o contraria ai dati di fatto, oppure temporale fra essi. Ogni volta che il suo antecedente vale FALSE, un condizionale materiale vale TRUE.
In italiano gli enunciati condizionali si possono esprimere in diversi modi. Abbiamo già visto le forme seguenti:

Se **P**, **Q**.
Se **P**, allora **Q**.

Altre forme di enunciati condizionali possono essere:

Q, se **P**.
Q, purché **P**.
Quando **P**, **Q**.

Per rappresentare simbolicamente gli enunciati condizionali useremo il simbolo $\rightarrow$. Dunque, per rappresentare simbolicamente un enunciato come 'Se stasera la luna ha l'alone, domani pioverà', posto

A = ‘Stasera la luna ha l’alone.’
B = ‘Domani pioverà.’

scriveremo:

(A→B)

In generale, dati due enunciati qualsiasi **P** e **Q**,

(P→Q)

è un enunciato condizionale. Ovviamente, anche ‘(**Q**→**P**)’ è un enunciato condizionale, ma distinto dal precedente.

LA FUNZIONE DI VERITÀ FCND

Al connettivo proposizionale → corrisponde una funzione di verità che chiameremo FCND. Questa funzione descrive il modo in cui il valore di verità di un condizionale dipende dai valori di verità dell’antecedente e del conseguente. La sua descrizione estensionale è:

FCND(FALSE, FALSE) = TRUE
FCND(FALSE, TRUE) = TRUE
FCND(TRUE, FALSE) = FALSE
FCND(TRUE, TRUE) = TRUE

Il primo ingresso di FCND è il valore di verità dell’antecedente, mentre il secondo ingresso è il valore di verità del conseguente. Da questa descrizione di FCND risulta che l’uscita è TRUE se il valore di verità dell’antecedente è FALSE, oppure se il valore di verità del conseguente è TRUE (oppure se si verificano entrambi i casi). In altre parole, l’uscita di FCND è FALSE se (e solo se) il valore di verità dell’antecedente è TRUE e il valore di verità del conseguente è FALSE.
Come per gli altri connettivi proposizionali, descriviamo FCND anche mediante la tavola di verità:

V(**P**)	V(**Q**)	V(**P**→**Q**)
FALSE	FALSE	TRUE
FALSE	TRUE	TRUE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

In base a questa informazione sul comportamento della funzione FCND possia-

mo costruire degli algoritmi per calcolarla. Un diagramma di flusso per calcolare FCND è:

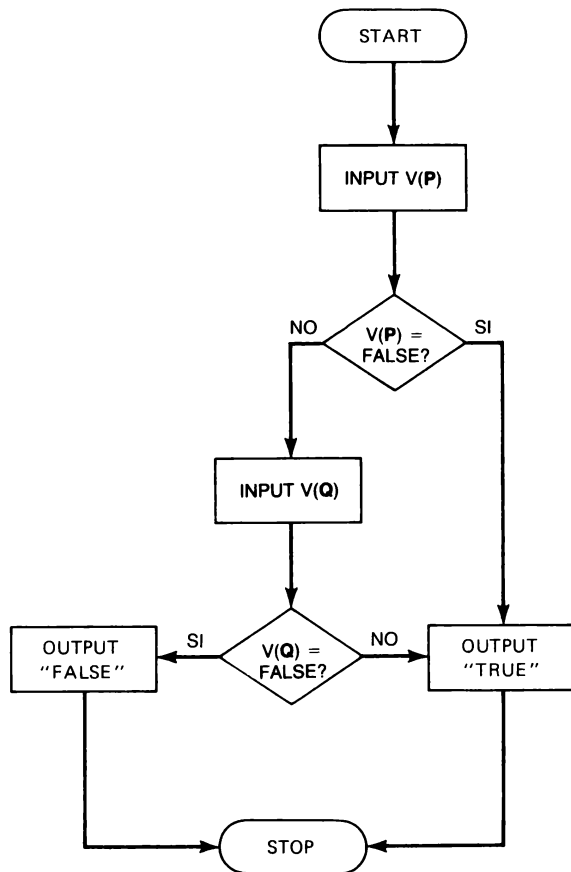



Figura 4.1 Diagramma di flusso per il calcolo di FCND

Un algoritmo descritto con il nostro pseudolinguaggio di programmazione è:

1. INPUT V(**P**).
2. INPUT V(**Q**).
3. IF V(**P**) = FALSE
   THEN (a) OUTPUT "TRUE."
   (b) STOP.
4. IF V(**Q**) = TRUE

THEN (a) OUTPUT "TRUE."
(b) STOP.
5. OUTPUT "FALSE."
6. STOP.

Ricorrendo alle già note associazioni aritmetiche, si trova che un'espressione aritmetica per calcolare FCND è:

MAX(1 − V(**P**), V(**Q**))

Considerando le quattro possibili coppie di valori di verità di **P** e **Q**, si calcoli MAX(1 − V(**P**), V(**Q**)) per verificare che questa espressione può effettivamente essere usata per calcolare FCND(V(**P**), V(**Q**)), conformemente alla tavola di verità.

RAPPRESENTAZIONE SIMBOLICA DEL CONDIZIONALE

Il trattamento dei condizionali richiede un po' più di attenzione rispetto al trattamento delle congiunzioni o delle disgiunzioni. Gli esempi di condizionali fin qui esaminati avevano tutti antecedente e conseguente atomici. Non è però sempre così: l'antecedente o il conseguente possono a loro volta essere enunciati molecolari. Si consideri, ad esempio:

Se ieri era venerdì oppure domani è Pasqua, allora oggi è sabato.

In questo caso, l'antecedente del condizionale è una disgiunzione:

Ieri era venerdì oppure domani è Pasqua.

Per evidenziare questo fatto, potremmo scrivere:

Se (ieri era venerdì oppure domani è Pasqua), allora oggi è sabato.

In modo analogo, nella nostra notazione evidenzieremo questo enunciato usando le parentesi. Posto:

A = 'Ieri era venerdì.'
B = 'Domani è Pasqua.'
C = 'Oggi è sabato.'

il condizionale sopra riportato si rappresenterà così:

$((A \vee B) \rightarrow C)$

È indispensabile che le parentesi siano messe nelle posizioni indicate. Se infatti avessimo scritto

$(A \vee (B \rightarrow C))$

avremmo rappresentato un enunciato completamente diverso, cioè:

O ieri era venerdì, oppure (se domani è Pasqua, allora oggi è sabato).

Si valutino i valori di verità di

$((A \vee B) \rightarrow C)$

e di

$(A \vee (B \rightarrow C))$

in corrispondenza di diversi valori di verità di A, B e C, per verificare che questi due enunciati diversi hanno, in alcuni casi, valori di verità diversi. Per tale verifica si può usare qualunque metodo di valutazione: le tavole di verità con TRUE e FALSE, le tavole di verità con 0 e 1, oppure gli algoritmi.
A volte è problematico decidere dove porre le parentesi nella rappresentazione simbolica di un enunciato in italiano contenente 'se ... allora ...'. Qui possiamo fornire solo qualche suggerimento generale. Se l'enunciato comincia con 'se', e se ciò che segue la parola 'allora' (o la virgola) viene pronunciato tutto d'un fiato, allora, nell'ipotesi che l'enunciato abbia la struttura

Se **P**, allora **Q**.

la sua rappresentazione simbolica corretta è probabilmente

$(\mathbf{P} \rightarrow \mathbf{Q})$

Vediamo alcuni esempi:

Enunciato:
Se l'oro è prezioso e trasportabile, allora bisogna nasconderlo o custodirlo al sicuro.
Rappresentazione simbolica:
$((A \wedge B) \rightarrow (C \vee D))$

Enunciato:
Se i nemici di una nazione sono potenti, allora, se hanno la volontà di usare la propria potenza, possono alterare l'equilibrio politico-militare.
Rappresentazione simbolica:
$(E \rightarrow (F \rightarrow G))$

Tuttavia, quando l’enunciato non comincia con ‘se’, o quando il conseguente, ossia ciò che segue ‘allora’, è interrotto da una virgola o da una pausa nella pronuncia, può darsi che il condizionale sia inserito entro un altro enunciato. Vediamo alcuni esempi:

Enunciato:
Non è affatto vero che, se io prenderò una scorciatoia attraverso i prati, allora tutti mi seguiranno.
Rappresentazione simbolica:
$\neg(H \rightarrow I)$

Enunciato:
Se pagheremo le tasse avremo meno risparmi, ma se non le pagheremo avremo paura di essere scoperti.
Rappresentazione simbolica:
$((A \rightarrow B) \wedge (\neg A \rightarrow C))$

SOLO SE

Un condizionale può anche essere formulato mediante l’espressione ‘solo se’. Anche l’enunciato

1. La combustione ha luogo solo se c’è l’ossigeno.

è condizionale. Ma qual è l’antecedente? E qual è il conseguente? Fra le due rappresentazioni possibili

2. La combustione ha luogo → c’è ossigeno.
3. C’è ossigeno → la combustione ha luogo.

ci si dovrebbe render conto che quella corretta è la prima. Dunque, l’enunciato che segue ‘solo se’ è il *conseguente* dell’enunciato condizionale (1). Come regola generale, si sostituisce direttamente ‘solo se’ con → nella rappresentazione simbolica dell’enunciato in italiano.
Un altro esempio è:

Darò l’esame solo se mio fratello starà meglio.

Quando la persona che ha detto ciò andrà a dare l’esame, si saprà che suo fratello sta meglio, perché, se così non fosse, tale persona non darebbe l’esame.

CONDIZIONI NECESSARIE E CONDIZIONI SUFFICIENTI

Il fatto che la combustione ha luogo solo in presenza di ossigeno si può esprimere anche così:

> La presenza di ossigeno è una condizione necessaria perché la combustione abbia luogo.

Ciò significa che *se* la combustione ha luogo *allora* l'ossigeno è presente, giacché se l'ossigeno *non* è presente la combustione *non* ha luogo. Dunque **P** è una *condizione necessaria* per **Q** quando l'assenza di **P** garantisce l'assenza di **Q**. (Si noti che 'la presenza di ossigeno' non è un enunciato. Tuttavia, tali espressioni sono facilmente esprimibili sotto forma di enunciati, e viceversa. Per comodità useremo quindi la stessa notazione in entrambi i casi.)
Diremo invece che **P** è una *condizione sufficiente* per **Q** quando la presenza di **P** garantisce la presenza di **Q**. Quando **P** è una condizione sufficiente per **Q** scriviamo 'Se **P**, allora **Q**'. Per esempio,

> L'arresto del flusso sanguigno per un'ora è una condizione sufficiente per la morte di un essere umano.

può essere espresso nella forma:

> Se il sangue di una persona cessa di scorrere per un'ora, allora tale persona muore.

Quando invece **P** è una condizione necessaria per **Q** scriviamo 'Se **Q**, allora **P**'. Per esempio,

> La respirazione è una condizione necessaria per rimanere in vita.

può essere espresso nella forma

> Se uno è vivo, allora respira.

Si noti che

> **P** è una condizione sufficiente per **Q**.

ha lo stesso significato di

> **Q** è una condizione necessaria per **P**.

e che entrambe le formulazioni equivalgono a

> Se **P**, allora **Q**.

4.2 L'ENUNCIATO BICONDIZIONALE

Accade a volte che **P** sia una condizione sia necessaria che sufficiente per **Q**. In questo caso si dirà:

Se **Q** allora **P**, e se **P** allora **Q**.

o, equivalentemente:

Se **P** allora **Q**, e se **Q** allora **P**.

Questa espressione potrebbe essere rappresentata simbolicamente così:

$((\mathbf{P} \rightarrow \mathbf{Q}) \wedge (\mathbf{Q} \rightarrow \mathbf{P}))$

e di solito si legge:

P se e solo se **Q**

In teoria, tutto ciò non presenta novità. Possiamo rappresentare gli enunciati contenenti 'se e solo se' usando, come qui, $\wedge$ e $\rightarrow$; possiamo poi determinare il valore di verità di ogni enunciato di questo tipo usando con attenzione FCND e FCNJ. Poiché però l'espressione 'se e solo se' è di uso frequente in logica, la chiameremo connettivo *bicondizionale* e useremo un simbolo apposito, la doppia freccia $\leftrightarrow$. Dunque '**P** se e solo se **Q**' si scriverà così:

$(\mathbf{P} \leftrightarrow \mathbf{Q})$

Esistono altri modi di esprimere il bicondizionale in italiano. Nei testi di logica e di matematica, al posto di 'se e solo se', si usa spesso l'abbreviazione 'sse'. Altre espressioni talvolta usate sono 'esattamente quando' ed 'esclusivamente quando'. Quando si trova una frase avente la forma '**P** se **Q**' o '**P** solo se **Q**', bisogna fare attenzione, perché spesso l'autore intende, in realtà, '**P** se e solo se **Q**', anche se non l'ha scritto esplicitamente. Viceversa, quando si trovi una frase avente la forma '**P** se e solo se **Q**', conviene accertarsi che si intenda sia 'se **P** allora **Q**', sia 'se **Q** allora **P**'.
Un algoritmo per calcolare '$(\mathbf{P} \leftrightarrow \mathbf{Q})$', inteso come '$((\mathbf{P} \rightarrow \mathbf{Q}) \wedge (\mathbf{Q} \rightarrow \mathbf{P}))$', potrebbe essere:

1. INPUT V(**P**).
2. INPUT V(**Q**).
3. LET V1 = FCND (V(**P**), V(**Q**)).
   (cioè V1 = V(**P**→**Q**).)
4. LET V2 = FCND (V(**Q**), V(**P**)).
   (cioè V2 = V(**Q**→**P**).)
5. OUTPUT FCNJ (V1, V2).
6. STOP.

Seguendo questo algoritmo si genera la seguente descrizione estensionale della funzione di verità, che chiameremo FBIC:

FBIC(FALSE, FALSE) = TRUE
FBIC(FALSE, TRUE) = FALSE
FBIC(TRUE, FALSE) = FALSE
FBIC(TRUE, TRUE) = TRUE

Esaminando ora la dipendenza del valore di verità di '(**P**↔**Q**)' dai valori di verità di **P** e **Q**, si vede che V(**P**↔**Q**) = TRUE se e solo se V(**P**) = V(**Q**). In altre parole, se V(**P**) = V(**Q**) allora V(**P**↔**Q**) = TRUE, e se V(**P**) ≠ V(**Q**) allora V(**P**↔**Q**) = FALSE. In base a questa relazione e al fatto che sottraendo un numero a se stesso si ottiene zero, possiamo costruire un algoritmo aritmetico per calcolare FBIC. A tal fine useremo la funzione *valore assoluto*, ABS, che si può definire nel modo seguente:

$\mathrm{ABS}(n) = n \quad \text{se } n \geqslant 0$
$\mathrm{ABS}(n) = -n \quad \text{se } n < 0$

Ad esempio, ABS(5) = 5, ABS(0) = 0, ABS(−5) = 5. ABS serve per determinare se V(**P**) = V(**Q**), giacché, rappresentando TRUE con 1 e FALSE con 0, risulta V(**P**) ≠ V(**Q**) sse ABS(V(**P**) − V(**Q**)) = 1 e V(**P**) = V(**Q**) sse ABS(V(**P**) − V(**Q**)) = 0. Ma siccome V(**P**↔**Q**) = 1 sse V(**P**) = V(**Q**) e V(**P**↔**Q**) = 0 sse V(**P**) ≠ V(**Q**), dobbiamo "invertire" l'uscita di ABS. L'algoritmo per calcolare FBIC è perciò:

1. INPUT V(**P**).
2. INPUT V(**Q**).
3. OUTPUT 1 − ABS (V(**P**) − V(**Q**)).
4. STOP.

4.3 LA DISGIUNZIONE ESCLUSIVA

Nel Capitolo 3 abbiamo detto che la disgiunzione può essere intesa in due sensi: *inclusivo* ed *esclusivo*. Ricordiamo che il valore di verità di una disgiunzione inclusiva si calcola con la funzione di verità FDSJ, la cui tavola di verità è:

V(**P**)	V(**Q**)	V(**P**∨**Q**)
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

Questa disgiunzione si dice inclusiva perché include la possibilità che entrambi i disgiunti abbiano valore di verità TRUE. Ad esempio, si supponga che a una persona venga chiesto come vanno gli studi e che essa risponda "Darò Fisica o Geometria". Nessuno l'accuserà di essere scorretta se in realtà darà entrambi gli esami. È certamente possibile darli entrambi: se quella persona pensa di darne almeno uno, nella frase 'Darò Fisica o Geometria' usa la parola 'o' in senso inclusivo. Spesso tuttavia si vuole escludere la possibilità che entrambi i disgiunti abbiano valore di verità TRUE. A volte questa possibilità è automaticamente esclusa: se una persona a cui chiediamo la data di oggi risponde "È il 21 o il 22", sappiamo che oggi non può essere sia il 21 che il 22. Questo è un esempio di disgiunzione esclusiva.
Altre volte, tuttavia, non è chiaro se una disgiunzione espressa in italiano sia esclusiva o inclusiva. Ad esempio, se un professore consiglia di inserire nel piano di studi il corso di Controllo di Processi o il corso di Robotica Industriale, ciò può significare che si deve scegliere uno solo dei due corsi. Potrebbe però significare anche che entrambi i corsi possono essere inseriti nel piano di studi. Nel primo caso, il consiglio del professore è una disgiunzione esclusiva; nel secondo caso, invece, è una disgiunzione inclusiva. Avendo solo le informazioni contenute in questo esempio, non si può sapere di quale tipo di disgiunzione si tratti. In una situazione reale, se non è immediatamente evidente di quale caso si tratti, occorrono ulteriori informazioni per decidere. A volte, per specificare il senso inclusivo, in italiano si usa l'espressione 'e/o'. Analogamente si può aggiungere l'espressione 'ma non entrambi' (o simili) per specificare il senso esclusivo. In altri casi, la strategia migliore consiste nell'assumere che si intenda il senso inclusivo, a meno che non si abbiano elementi per ritenere il contrario.
Esistono diversi simboli per rappresentare la disgiunzione esclusiva; noi useremo semplicemente l'abbreviazione XOR (per "eXclusive OR", cioè "'o' esclusivo").
Dunque

(**P** XOR **Q**)

sarà la rappresentazione simbolica di un enunciato del tipo

P o **Q**, ma non entrambi.

Se due enunciati sono collegati da una disgiunzione esclusiva, ciò significa che uno e uno solo di essi ha valore di verità TRUE. Per la disgiunzione esclusiva useremo la funzione di verità FXOR:

V(**P** XOR **Q**) = FXOR(V(**P**), V(**Q**))

Questa funzione può essere descritta estensionalmente nel modo seguente:

FXOR(FALSE, FALSE) = FALSE
FXOR(FALSE, TRUE) = TRUE

FXOR(TRUE, FALSE) = TRUE
FXOR(TRUE, TRUE) = FALSE

Si consideri ad esempio l'enunciato

Oggi è il 21 o il 22.

Posto

A = 'Oggi è il 21.' B = 'Oggi è il 22.'

tale enunciato può essere rappresentato simbolicamente così:

(A XOR B)

Se V(A) e V(B) sono entrambi FALSE, allora '(A XOR B)' deve sicuramente avere valore di verità FALSE. Se invece V(A) è TRUE, V(B) deve essere FALSE, dunque '(A XOR B)' ha valore di verità TRUE. Analogamente, se V(B) è TRUE e V(A) no, '(A XOR B)' ha valore di verità TRUE. Nel caso di questo enunciato, non ha molto senso supporre che V(A) e V(B) siano entrambi TRUE, ma se in qualche modo lo fossero, '(A XOR B)' avrebbe effettivamente valore di verità FALSE. Un esempio più plausibile per illustrare quest'ultima possibilità potrebbe essere il seguente: si supponga che un professore dica, all'inizio del corso, che l'esame può essere superato o con un compitino, o con un colloquio orale. Se qualcuno poi riesce a fare sia il compitino sia l'orale, ne risulta che ciò che il professore aveva detto non era vero.
Per calcolare FXOR si può scrivere un algoritmo che usa la funzione aritmetica ABS, già utilizzata nell'algoritmo per il calcolo di FBIC. Anche in questo caso si sfrutta il fatto che V(**P**) = V(**Q**) se e solo se ABS(V(**P**) − V(**Q**)) = 0:

1. INPUT V(**P**).
2. INPUT V(**Q**).
3. OUTPUT ABS (V(**P**) − V(**Q**)).
4. STOP.

Poiché

(**P** XOR **Q**)

significa

1. **P** o **Q**, ma non entrambi.

si può esprimere XOR in termini di congiunzione, negazione e disgiunzione inclusiva. Ricordando che 'ma' corrisponde a ∧ e che 'entrambi' significa '**P** e **Q**', scriveremo la (1) così:

$((\mathbf{P} \vee \mathbf{Q}) \wedge \neg (\mathbf{P} \wedge \mathbf{Q}))$

4.4 A MENO CHE

In italiano corrente diverse proposizioni vengono espresse mediante enunciati che contengono il connettivo 'a meno che'. Come vedremo in questo paragrafo, 'a meno che' non è, in realtà, un nuovo connettivo, perché può essere espresso in termini dei connettivi che conosciamo già. Si consideri l'enunciato

1. Darò l'esame, a meno che mia sorella non sia ammalata.

L'enunciato (1) significa che se mia sorella non sarà ammalata, allora darò l'esame. Posto

A = 'Darò l'esame.'
B = 'Mia sorella sarà ammalata.'

la (1) si può rappresentare simbolicamente così:

2. $(\neg B \rightarrow A)$

Si noti che l'enunciato (1) potrebbe essere interpretato anche nel modo seguente: se non darò l'esame, vorrà dire che mia sorella è ammalata. Simbolicamente ciò si rappresenterebbe così:

3. $(\neg A \rightarrow B)$

Fortunatamente si può dimostrare che queste due rappresentazioni valgono TRUE oppure FALSE esattamente nelle stesse circostanze. Dunque sia (2) che (3) sono rappresentazioni simboliche appropriate di (1).
Ci si potrebbe chiedere perché non abbiamo rappresentato (1) come

4. $(B \rightarrow \neg A)$

ossia: se mia sorella sarà ammalata, allora non darò l'esame. Spesso però con un enunciato come (1) si intende indicare che cosa si farà se la propria sorella non sarà ammalata, riservandosi di decidere come comportarsi nel caso in cui la sorella sia ammalata. Dopo tutto, si potrebbe ugualmente dare l'esame.
Chiameremo "debole" l'interpretazione dell'enunciato (1) rappresentata da (2). Un enunciato contenente 'a meno che' comporta sempre almeno l'interpretazione debole. In alcuni contesti esso può avere anche il significato (4). Chiameremo "forte" l'interpretazione secondo cui (1) significa sia (2) che (4). La scelta dell'interpretazione appropriata dipende dal contesto, proprio come la scelta fra $\vee$ e XOR per rappresentare la parola 'o'. In effetti, come vedremo tra breve, c'è un preciso parallelismo fra queste due scelte.
Come ulteriore esempio, si consideri

5. Ceneremo alle 8, a meno che non andiamo al cinema.

Posto

E = 'Ceneremo alle 8.'
F = 'Andremo al cinema.'

l'interpretazione debole di (5) è

6. $(\neg F \rightarrow E)$

Si supponga però che dal contesto si possa ricavare chiaramente che il significato di (5) è:

7. Ceneremo alle 8, a meno che non andiamo al cinema, nel qual caso ceneremo alle 7.

È ragionevole supporre che, se ceneremo alle 7, non ceneremo alle 8. Perciò, in questo contesto, si deve intendere che (5) significhi che se non andremo al cinema ceneremo alle 8, mentre se andremo al cinema non ceneremo alle 8. In altre parole, ceneremo alle 8 se e solo se non andremo al cinema (interpretazione forte):

8. $(E \leftrightarrow \neg F)$

Con qualche semplice calcolo sulle tavole di verità, si può vedere che l'interpretazione debole di (5) può essere espressa dalla disgiunzione inclusiva

9. $(E \vee F)$

e che l'interpretazione forte può essere espressa dalla disgiunzione esclusiva

10. (E XOR F)

4.5 NOR

Un altro connettivo di uso frequente in italiano è espresso dalla parola 'né', come nell'enunciato

1. Né Anna né Roberto sono venuti alla festa.

Ciò significa semplicemente che Anna non è venuta alla festa e Roberto non è venuto alla festa. Si supponga che qualcuno chieda se Anna *o* Roberto sono venuti alla festa. Se (1) è vero, la risposta è no; in altre parole, (1) è la negazione di

Anna o Roberto sono venuti alla festa.

Dunque un enunciato contenente 'né ... né ...' è la negazione di un enunciato disgiuntivo. Per questo motivo, un simbolo correntemente usato per rappresentare 'né' è il segno di disgiunzione attraversato da una barra verticale, cioè una freccia rivolta verso il basso: $\downarrow$. Per semplicità useremo però il trigramma NOR (in inglese "né"). Con evidente significato delle abbreviazioni, (1) si può rappresentare simbolicamente così:

(A NOR B)

Il valore di verità di un enunciato contenente NOR si calcola con la funzione FNOR, la cui descrizione estensionale, come si può facilmente verificare, è la seguente:

FNOR(FALSE, FALSE) = TRUE
FNOR(FALSE, TRUE) = FALSE
FNOR(TRUE, FALSE) = FALSE
FNOR(TRUE, TRUE) = FALSE

Si noti che V(**P** NOR **Q**) = V($\neg$(**P**$\vee$**Q**)).

4.6 NAND

Se NOR è la negazione della disgiunzione inclusiva, ci si può chiedere quale sia la negazione della congiunzione. Purtroppo non esiste in italiano una parola che la esprima, ma in informatica è stata coniata la parola 'NAND' (per "Not-AND", cioè "non-è"). Dunque

(**P** NAND **Q**)

significa:

$\neg$(**P**$\wedge$**Q**)

Esiste anche un altro simbolo di uso corrente per NAND, la barra verticale: |. Noi useremo però il simbolo NAND, sia perché è di uso più frequente in informatica, sia perché in alcuni linguaggi di programmazione la barra verticale rappresenta invece la disgiunzione. Poiché NAND si può esprimere in termini di $\neg$ e di $\wedge$, per la descrizione estensionale e algoritmica della funzione di verità FNAND si rimanda agli esercizi (in particolare all'Esercizio F di questo capitolo).

4.7 I CONNETTIVI VEROFUNZIONALI BINARI

Dobbiamo ora aggiungere nuove regole per definire gli enunciati:

5. Se le stringhe **P** e **Q** sono enunciati,
   allora il condizionale (**P**→**Q**) è un enunciato.
6. Se le stringhe **P** e **Q** sono enunciati,
   allora il bicondizionale (**P**↔**Q**) è un enunciato.
7. Se le stringhe **P** e **Q** sono enunciati,
   allora la loro disgiunzione esclusiva (**P** XOR **Q**) è un enunciato.
8. Se le stringhe **P** e **Q** sono enunciati,
   allora (**P** NOR **Q**) è un enunciato.
9. Se le stringhe **P** e **Q** sono enunciati,
   allora (**P** NAND **Q**) è un enunciato.

Finora, senza contare 'a meno che', abbiamo considerato sette connettivi verofunzionali binari: ∧, ∨, →, ↔, XOR, NOR, NAND. A questo punto è logico chiedersi: quanti connettivi verofunzionali binari esistono? Sussistono delle relazioni significative fra essi?
Per rispondere a queste domande conviene elencare sistematicamente tutti i possibili connettivi. A tal fine, si considerino le caratteristiche di una funzione binaria di verità: essa deve avere una uscita, di valore 0 o 1 (cioè FALSE o TRUE), per ogni possibile coppia di valori di verità in ingresso. Poiché tali coppie sono solo quattro (ossia 00, 01, 10, 11), tutte le possibili uscite si possono elencare come in Tabella 4.1.
Noi abbiamo discusso solo le sette funzioni di verità a cui abbiamo attribuito un nome cominciante per 'F', ma tutte le sedici funzioni descritte estensionalmente nella tabella possono essere trattate in modo analogo. Esistono cioè algoritmi per calcolarle e funzioni aritmetiche utilizzabili in tali algoritmi, nonché modi per esprimerle in italiano (anche se in alcuni casi ciò può richiedere una certa abilità).

4.8 CONCLUSIONI

In questo capitolo abbiamo completato l'introduzione ai connettivi verofunzionali binari. Abbiamo considerato le descrizioni estensionali delle corrispondenti funzioni di verità e abbiamo presentato degli algoritmi per calcolarle.
Abbiamo trattato i *condizionali*, occupandoci soprattutto degli *enunciati condizionali materiali* ('Se **P**, allora **Q**') e dei modi in cui possono essere espressi in italiano, inclusi i concetti di *condizioni necessarie* e *condizioni sufficienti*. Abbiamo poi considerato il *bicondizionale* ('**P** se e solo se **Q**'), la *disgiunzione esclusiva* XOR ('**P** o **Q**, ma non entrambi') e i connettivi NOR e NAND. Abbiamo anche considerato il connettivo 'a meno che'. Infine, abbiamo elencato tutte le sedici funzioni binarie di verità.

Tabella 4.1 Le funzioni binarie di verità

n°	V(P) 0 V(Q) 0	V(P) 0 V(Q) 1 (Ingressi)	V(P) 1 V(Q) 0	V(P) 1 V(Q) 1	Nome	Funzione	Simbolo	Corrispondente italiano
		Uscite						
0	0	0	0	0	(nessuno)	(nessuna)	(nessuno)	(nessuno)
1	0	0	0	1	congiunzione	FCNJ	$\wedge$	**P** e **Q**
2	0	0	1	0	non-condizionale materiale*	(nessuna)	$\nrightarrow$	**P** ma non **Q**
3	0	0	1	1	(nessuno)	(nessuna)	(nessuno)	(nessuno)
4	0	1	0	0	non-condizionale converso*	(nessuna)	$\nleftarrow$	**Q** ma non **P**
5	0	1	0	1	(nessuno)	(nessuna)	(nessuno)	(nessuno)
6	0	1	1	0	disgiunzione esclusiva	FXOR	XOR	**P** o **Q**, ma non entrambi
7	0	1	1	1	disgiunzione inclusiva	FDSJ	$\vee$	**P** o **Q**
8	1	0	0	0	né	FNOR	NOR	né **P** né **Q**
9	1	0	0	1	bicondizionale	FBIC	$\leftrightarrow$	**P** se e solo se **Q**
10	1	0	1	0	(nessuno)	(nessuna)	(nessuno)	(nessuno)
11	1	0	1	1	condizionale converso*	(nessuna)	$\leftarrow$	**P** se **Q**
12	1	1	0	0	(nessuno)	(nessuna)	(nessuno)	(nessuno)
13	1	1	0	1	condizionale materiale	FCND	$\rightarrow$	se **P**, allora **Q**
14	1	1	1	0	nand	FNAND	NAND	non insieme **P** e **Q**
15	1	1	1	1	(nessuno)	(nessuna)	(nessuno)	(nessuno)

*nomi proposti da Church (1956)

4.9 ESERCIZI

A. Rappresentare simbolicamente gli enunciati sotto riportati, usando le abbreviazioni seguenti:

A = 'Arturo è innocente.' B = 'Barbara è innocente.'
C = 'Carlo è innocente.' D = 'Carlo dice la verità.'

1. Se Arturo è innocente, allora lo è anche Barbara.
2. Carlo è innocente solo se lo è anche Arturo.
3. Se né Arturo né Barbara sono innocenti, allora Carlo è innocente.
4. Solo se Carlo dice la verità Arturo è innocente.
5. A meno che Carlo non dica la verità, Barbara non è innocente.
6. Arturo è innocente, a meno che Carlo non menta.
7. Barbara è innocente, nel caso in cui Carlo dica la verità.
8. Se Barbara e Carlo sono innocenti, allora Arturo non lo è.
9. Se, ma solo se, Carlo dice la verità, allora Arturo è innocente.
10. Barbara è innocente se e solo se Arturo è innocente e Carlo dice la verità.
11. Carlo è innocente se lo sono Barbara o Arturo.
12. Carlo dice la verità se e solo se è innocente.
13. Carlo dice la verità ed è innocente se e solo se Arturo è innocente.
14. Carlo è innocente esclusivamente nel caso in cui dice la verità.
15. Se Carlo dice la verità, Arturo e Barbara sono innocenti.
16. Arturo è innocente se e solo se non si dà il caso che Barbara e Carlo siano entrambi innocenti.

B. Rappresentare simbolicamente gli enunciati seguenti, introducendo abbreviazioni appropriate per gli enunciati atomici.

1. Se vuoi prendere un bel voto, ci riuscirai se studierai molto.
2. Se vuoi prendere un bel voto e se studierai molto, ci riuscirai.
3. Se vuoi prendere un bel voto, ci riuscirai solo se studierai molto.
4. A meno che non abbiate una carta di credito, potete noleggiare la macchina se e solo se lasciate una cauzione di 200 000 lire.
5. Non siete responsabili di eventuali danni, a meno che, ovviamente, l'incidente non sia avvenuto per colpa vostra, nel qual caso la compagnia pagherà se e solo se pagherete le prime 200 000 lire.
6. Non si può speculare in Borsa con successo se non si è fortunati, a meno che non si abbiano dei validi informatori.

C. Calcolare il valore di verità di ciascuno degli enunciati sotto riportati, usando i seguenti valori di verità per gli enunciati atomici:

V(A) = TRUE
V(B) = TRUE
V(C) = FALSE
V(D) = FALSE

1. $(A \rightarrow (B \vee C))$
2. $(B \rightarrow (A \wedge D))$
3. $((A \vee C) \rightarrow D)$

4. $((C \lor D) \rightarrow (A \lor C))$
5. $(A \rightarrow (B \rightarrow \neg C))$
6. $((C \land D) \rightarrow \neg A)$
7. $(B \rightarrow \neg (A \lor C))$
8. $(A \rightarrow (D \rightarrow \neg (B \land C)))$
9. $((B \rightarrow (A \lor C)) \rightarrow (D \land B))$
10. $((B \land D) \rightarrow (C \lor (A \land B)))$
11. $(A \leftrightarrow (B \lor C))$
12. $(C \leftrightarrow (A \leftrightarrow D))$
13. $(B \leftrightarrow (C \text{ NOR } D))$
14. $(C \leftrightarrow \neg (A \text{ NAND } D))$
15. $(\neg (A \land C) \leftrightarrow (A \text{ NAND } C))$
16. $((C \lor (A \leftrightarrow D)) \rightarrow (A \rightarrow C))$
17. $(\neg A \leftrightarrow (A \text{ NOR } A))$
18. $((B \text{ NAND } B) \leftrightarrow (B \text{ NOR } B))$
19. $((C \text{ NOR } C) \leftrightarrow (C \text{ NAND } C))$
20. $((A \text{ NAND } A) \leftrightarrow (\neg A \lor C))$
21. $((A \text{ XOR } B) \text{ XOR } C)$
22. $((A \text{ XOR } C) \text{ XOR } D)$
23. $((D \text{ XOR } A) \text{ NAND } C)$
24. $(C \text{ NOR } (A \text{ XOR } B))$
25. $\neg (A \text{ XOR } B)$
26. $(\neg A \text{ XOR } \neg B)$
27. $(\neg A \text{ XOR } B)$
28. $(A \text{ XOR } \neg B)$
29. $((A \text{ XOR } B) \land C)$
30. $(A \text{ XOR } (B \land C))$
31. $((A \text{ XOR } B) \land \neg (A \text{ XOR } B))$
32. $((\neg A \text{ XOR } B) \land (\neg B \text{ XOR } A))$

D. Disgiunzione esclusiva

1. Si consideri l’enunciato seguente:
   “Se un professore dice che nel piano di studi si può inserire il corso di Controllo di Processi o il corso di Robotica Industriale, ciò può significare che uno solo dei due corsi può essere inserito nel piano di studi, *oppure* che entrambi i corsi possono essere inseriti nel piano di studi.”
   La parola ‘oppure’ evidenziata in corsivo ha valore inclusivo o esclusivo? Perché?
2. Scrivere un algoritmo per calcolare FXOR mediante MAX e MIN.
3. Scrivere un algoritmo per calcolare FXOR usando FBIC.
4. Scrivere un algoritmo per calcolare FBIC usando FXOR.
5. Scrivere un algoritmo per calcolare FXOR, sfruttando il fatto che ‘(**P** XOR **Q**)’ equivale a ‘$((\mathbf{P} \lor \mathbf{Q}) \land \neg (\mathbf{P} \land \mathbf{Q}))$’.

E. NOR
  1. Scrivere un algoritmo per calcolare FNOR mediante FDSJ.
  2. Scrivere un algoritmo per valutare (**P** NOR **Q**) e un algoritmo per valutare $\neg$ (**P**∨**Q**) e verificare che, a parità di ingressi, le uscite sono uguali.

F. NAND
  1. Descrivere estensionalmente la funzione di verità FNAND.
  2. Scrivere un algoritmo che calcola FNAND usando un'opportuna funzione aritmetica.
  3. Mostrare come XOR può essere espresso in termini di $\neg$, ∧ e NAND. [*Suggerimento*: '(**P** NAND **Q**)' vale TRUE se al massimo uno dei due enunciati **P** e **Q** vale TRUE.]
  4. Scrivere un algoritmo che calcola FXOR mediante le funzioni aritmetiche usate per $\neg$, ∧ e NAND.

G. Per ciascuno dei connettivi 0, 3, 5, 10, 12 e 15 della Tabella 4.1, scrivere un enunciato in italiano che lo esemplifichi in modo appropriato.

5

Logica enunciativa: algoritmi per calcolare i valori di verità e per determinare se una formula è ben formata

Finora non abbiamo approfondito il problema del calcolo dei valori di verità di enunciati molecolari relativamente complessi, a partire dai valori di verità degli enunciati atomici che li costituiscono. Ci siamo invece limitati a considerare enunciati molecolari piuttosto semplici, che illustrassero le proprietà dei connettivi proposizionali fondamentali $\neg$, $\wedge$, $\vee$, e $\rightarrow$.
Dobbiamo però definire un algoritmo che permetta di calcolare i valori di verità di enunciati più complessi. Come vedremo più avanti, ciò risulterà molto utile per determinare se un'argomentazione in logica enunciativa è valida oppure no.
Per definire un algoritmo che calcoli il valore di verità di un enunciato molecolare seguiremo un metodo già noto. Dapprima considereremo il modo in cui noi stessi risolveremmo il problema (come già avevamo fatto per definire un algoritmo di addizione). Poi raffineremo il procedimento, in modo che chiunque (anche un calcolatore) possa eseguirlo, senza aver bisogno di nozioni sulla logica o sui valori di verità.

5.1 UN METODO INFORMALE PER IL CALCOLO DEI VALORI DI VERITÀ

Agli enunciati atomici A, B, C e D siano assegnati i seguenti valori di verità:

$V(A) = 1$
$V(B) = 0$
$V(C) = 1$
$V(D) = 0$

ove, come nel seguito di questo capitolo e nel capitolo successivo, TRUE e FALSE vengono rappresentati per comodità da 1 e 0. Si consideri ora un enunciato

molecolare costituito da questi enunciati atomici; ad esempio:

$(A \rightarrow (\neg B \wedge (C \vee D)))$

Qual è il valore di verità di questo enunciato molecolare, una volta che siano assegnati i valori di verità degli enunciati atomici?
Per calcolare questo valore di verità, cominciamo col mettere i corrispondenti valori di verità al posto delle lettere A, B, C e D che rappresentano gli enunciati atomici:

$(1 \rightarrow (\neg 0 \wedge (1 \vee 0)))$

È importante notare che, in base alle regole viste, questo *non* è un enunciato: è una cosiddetta *formula ibrida*, costituita da valori di verità e connettivi proposizionali. In italiano non ha senso dire "Se TRUE, allora non FALSE e, insieme, o TRUE o FALSE"; vedremo però che si tratta di una formula davvero utile.
A questo punto come procediamo? In questa formula cerchiamo qualcosa che sia già noto dai capitoli precedenti, cioè semplici combinazioni di valori di verità e connettivi proposizionali.
Due combinazioni di questo tipo saltano subito all'occhio:

$\neg 0$

e

$(1 \vee 0)$

Diremo che queste sono *sottoformule* della formula in esame, perché sono parti di tale formula e hanno la struttura degli enunciati molecolari semplici visti negli ultimi due capitoli.
La prima sottoformula, '$\neg 0$', può essere sostituita dal valore di verità 1, giacché la funzione FNEG associata alla negazione "inverte" il valore di verità dell'enunciato cui è applicata. Possiamo cioè sostituire '$\neg 0$' con l'uscita di FNEG(0), ossia con 1. Analogamente la sottoformula '$(1 \vee 0)$' può essere sostituita dal valore di verità 1. Per rendersene conto basta consultare la tavola di verità di $\vee$, oppure applicare FDSJ ai valori di verità 0 e 1. In modo informale, basta ricordare che una disgiunzione vale TRUE se almeno uno dei disgiunti vale TRUE. Perciò si può sostituire '$(1 \vee 0)$' con il valore di verità 1.
Scrivendo questi passaggi si ha:

$(A \rightarrow (\neg B \wedge (C \vee D)))$
$(1 \rightarrow (\neg 0 \wedge (1 \vee 0)))$
$(1 \rightarrow (1 \quad \wedge (1 \vee 0)))$ [si sostituisce '$\neg 0$' con 1]
$(1 \rightarrow (1 \quad \wedge \quad 1 \quad))$ [si sostituisce '$(1 \vee 0)$' con 1]

Il prossimo passaggio consiste nell'applicare ancora, alle parti più semplici di quest'ultima formula, le nozioni già apprese riguardo ai valori di verità e ai connettivi proposizionali. Riconosciamo la sottoformula '$(1 \wedge 1)$', che, in virtù delle proprietà della congiunzione, può essere sostituita dal valore di verità 1, come si può determinare esaminando la tavola di verità di $\wedge$ o applicando la funzione FCNJ. I passaggi successivi sono perciò:

$$(1 \rightarrow (1 \wedge 1))$$
$$(1 \rightarrow \quad 1 \quad)$$
$$1$$

L'ultima riga, costituita da un solo valore di verità, è stata ottenuta in virtù del fatto che la formula ibrida '$(1 \rightarrow 1)$' può essere sostituita dal valore di verità 1, come si può verificare consultando la tavola di verità di $\rightarrow$ o applicando FCND. L'ultima riga indica che il valore di verità dell'enunciato molecolare originario è 1, cioè TRUE, in base ai valori di verità che erano stati assegnati agli enunciati atomici.

OSSERVAZIONI SUL METODO INFORMALE

È opportuno fare alcune considerazioni su quanto è appena stato esposto. In primo luogo, si noti che, per determinare il valore di verità di un enunciato molecolare, basta guardare l'ultima riga: non occorre considerare i passaggi che hanno condotto ad essa, a meno che non se ne voglia verificare la correttezza. In secondo luogo, ad ogni passaggio successivo alla sostituzione degli enunciati atomici con i loro valori di verità, si cercano le parti della formula a cui si possono applicare direttamente le funzioni di verità che calcolano $\neg$, $\wedge$, $\vee$, e $\rightarrow$, come, ad esempio, '$\neg 1$', '$(0 \wedge 0)$', '$(1 \vee 0)$', '$(1 \rightarrow 1)$'. Una volta individuate queste sottoformule, si sostituisce ciascuna di esse (comprese le eventuali parentesi immediatamente adiacenti) con il valore di verità fornito dall'opportuna funzione di verità. Il processo termina quando si rimane con un solo valore di verità. Come ulteriore esempio, si considerino l'enunciato molecolare

$$(\neg A \wedge (A \rightarrow ((C \vee D) \rightarrow D)))$$

e i seguenti assegnamenti di valori di verità agli enunciati atomici:

$V(A) = 0$	$V(C) = 1$
$V(B) = 1$	$V(D) = 0$

Per determinare il valore di verità dell'enunciato molecolare si procede nel modo seguente (le spiegazioni sono omesse):

```
(¬A∧(A→((C∨D)→ D)))
(¬ 0∧ (0→((1 ∨0) → 0)))
(1   ∧ (0→((1 ∨0) → 0)))
(1   ∧ (0→ ( 1    → 0)))
(1   ∧ (0→         0  ))
(1   ∧   1            )
     1
```

Consideriamo un ulltimo esempio, avendo cura di scrivere il valore di verità sostitutivo esattamente sotto il connettivo della sottoformula sostituita:

$$(B \vee (C \rightarrow ((D \wedge \neg D) \rightarrow A)))$$

con i valori di verità

$V(A) = 1$ $\qquad$ $V(C) = 1$
$V(B) = 0$ $\qquad$ $V(D) = 0$

Il calcolo si svolge nel modo seguente:

```
(B∨ (C→((D∧¬D)→A)))
(0∨ (1 →((0 ∧¬0 )→1 )))
(0∨ (1 →((0 ∧1   )→1 )))
(0∨ (1 →(  0      →1 )))
(0∨ (1 →           1  ))
(0∨   1               )
  1
```

SOTTOFORMULE PIÙ INTERNE

Per definire un algoritmo che calcoli il valore di verità di un enunciato molecolare, è necessario introdurre il concetto di *sottoformula più interna*, che, una volta precisato, permetterà di descrivere esplicitamente il procedimento di ricerca delle sottoformule a cui si possono applicare direttamente le funzioni di verità più semplici.

Dopo aver sostituito gli enunciati atomici con i loro valori di verità, e dopo aver sostituito '¬0' con '1' e '¬1' con '0', si ottiene una formula che non contiene simboli di negazione prefissi a parti atomiche. Data una tale formula, priva di parti atomiche negate, le sue *sottoformule più interne* (che possono essere più di una) sono le sottoformule racchiuse dal maggior numero di parentesi. Ad esempio, in

$$(1 \rightarrow ((0 \wedge (1 \vee 0)) \wedge 1))$$

la sottoformula più interna, che in questo caso è unica, è '(1 ∨ 0)'. È la parte di formula "più circondata" da parentesi: è racchiusa fra quattro livelli di parentesi, compresa la coppia di parentesi che fa parte della sottoformula stessa. Nella formula

((0→1)∨(1→0))

le sottoformule più interne sono '(0→1)' e '(1→0)', che sono entrambe racchiuse fra due sole coppie di parentesi. Si noti che la sottoformula più interna di '(1∨0)' è '(1∨0)' stessa.

5.2 L'ALGORITMO CALCOLO DEL VALORE DI VERITÀ

Usando il concetto di sottoformula più interna, possiamo definire l'algoritmo completo per il calcolo del valore di verità di un enunciato molecolare complesso:

ALGORITMO CALCOLO DEL VALORE DI VERITÀ

1. INPUT un enunciato **P**.
2. INPUT i valori di verità degli enunciati atomici di **P**.
3. Sostituire ogni enunciato atomico di **P** con il valore di verità ad esso assegnato, per generare una formula ibrida.
4. Leggendo la formula ibrida da sinistra a destra, cancellare tutte le coppie ¬ ¬ di simboli di negazione adiacenti.
5. WHILE la formula ha più di un carattere,
    - (a) Leggendo la formula da sinistra a destra, sostituire ogni '¬ 1' con '0'.
    - (b) Leggendo la formula da sinistra a destra, sostituire ogni '¬ 0' con '1'.
    - (c) IF rimane un solo valore di verità,
        - THEN
        - (i) OUTPUT tale valore.
        - (ii) STOP.
    - (d) Individuare le sottoformule più interne.
    - (e) FOR ogni sottoformula più interna:
        - (i) IF il simbolo centrale della sottoformula è ∧,
            - THEN
            - (1) Applicare FCNJ ai valori di verità contenuti nella sottoformula.
            - (2) Sostituire la sottoformula con l'uscita di FCNJ.
        - (ii)—(iv) Procedere analogamente per ∨, → e ↔.
6. OUTPUT il singolo valore di verità ottenuto.
7. STOP.

Il ciclo al passo 5 permette di ridurre la formula fino ad ottenere un solo valore di verità. Questo algoritmo descrive essenzialmente il procedimento informale appena visto per il calcolo del valore di verità di un enunciato molecolare. Si noti che la negazione, essendo un connettivo unario, crea qualche problema nella definizione dell'algoritmo e richiede dei passi appositi: 4, 5(a) e 5(b).
I passi 5(d) e 5(e) sono espressi ad un livello piuttosto astratto, quindi vanno raffinati.

RAFFINAMENTO DEL PASSO 5(d): INDIVIDUAZIONE DELLE SOTTOFORMULE PIÙ INTERNE

Come si è già visto, si tratta di determinare quali parti della formula si trovano al livello più interno di parentesi. Come si può risolvere questo problema in modo automatico?
Si supponga di numerare i caratteri della stringa diversi dagli spazi bianchi; ad esempio:

```
( ( 1 →0 )→0 )
1 2 3 4 5 6 7 8 9
```

Si immagini ora un contatore che percorra, da sinistra a destra, i caratteri numerati della stringa. Il contatore assume inizialmente il valore 0. Quando incontra una parentesi aperta, aggiunge 1 al proprio valore; quando incontra una parentesi chiusa, sottrae 1 al proprio valore; se invece il carattere incontrato non è una parentesi, il suo valore rimane invariato. Il contatore vale 0 prima di cominciare a scandire la stringa e *deve* valere 0 dopo averla scandita. Se ciò non avviene, significa che il numero delle parentesi aperte è diverso dal numero delle parentesi chiuse, e quindi la stringa originaria, non essendo conforme alle regole viste, non è un enunciato.
Si memorizza il massimo valore raggiunto dal contatore durante la scansione da sinistra a destra, poi si fa partire un secondo contatore, inizialmente di valore 0, che scandisce la stringa da sinistra a destra con le stesse regole: '(' aggiunge 1, ')' sottrae 1.

1. La prima sottoformula più interna comincia nella posizione in cui il secondo contatore assume per la prima volta il massimo valore del primo contatore.
2. Questa formula più interna termina nella prima posizione in cui il secondo contatore diventa minore di tale valore.

Siccome ci può essere più di una sottoformula più interna, il secondo contatore deve continuare a percorrere la stringa, memorizzando le posizioni in cui iniziano e finiscono le sottoformule più interne. Le posizioni così memorizzate possono poi essere usate per estrarre le sottoformule che verranno analizzate e valutate al successivo passo 5(e).

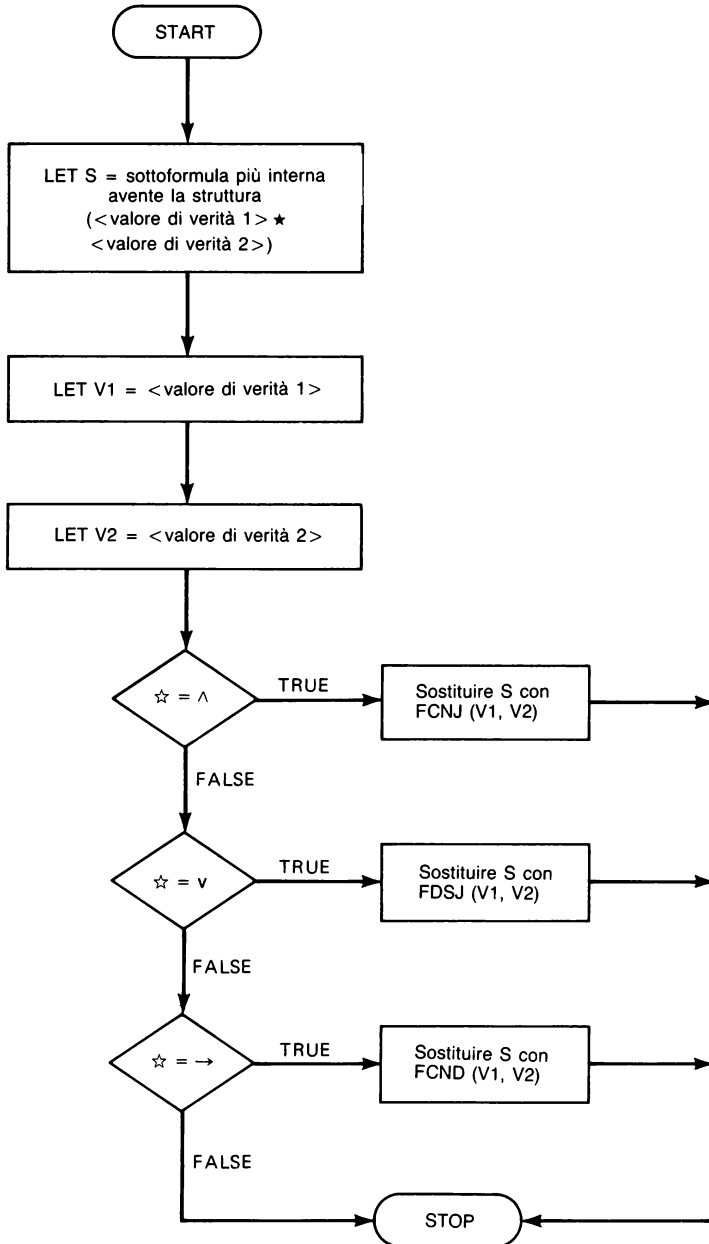



Figura 5.1 Diagramma di flusso per sostituire sottoformule più interne con valori di verità

Un esempio dovrebbe essere sufficiente per illustrare come funzionano questi due contatori.

Formula:		(	1	→	(	0	∧	(	0	∨	1	)	)	)
Posizione:		1	2	3	4	5	6	7	8	9	10	11	12	13
Primo contatore:	0	1	1	1	2	2	2	3	3	3	3	2	1	0

Il massimo valore del primo contatore è 3. Perciò il secondo contatore memorizzerà il fatto che la sottoformula più interna inizia nella posizione 7 e finisce nella posizione 11. (Alcune delle complicazioni introdotte dalle parentesi si possono evitare ricorrendo ad una notazione che ne sia priva, come la *notazione polacca*, che vedremo nel Capitolo 7.)

RAFFINAMENTO DEL PASSO 5(e): SOSTITUZIONE DELLE SOTTOFORMULE PIÙ INTERNE CON I VALORI DI VERITÀ

Ogni sottoformula più interna deve avere la struttura

(<valore di verità 1> ☆ <valore di verità 2>)

dove <valore di verità 1> e <valore di verità 2> valgono 0 o 1 e ☆ rappresenta un connettivo proposizionale binario, come ∧, ∨ o →. Il passo 5(e) indica il valore di verità con cui si deve sostituire l'intera sottoformula, parentesi comprese. Un diagramma di flusso del sottoprogramma del passo 5(e) è illustrato nella Figura 5.1. Ogni sottoformula più interna viene dunque sostituita con un valore di verità, conformemente alle funzioni di verità studiate nei capitoli precedenti.
Ad esempio:

(1∧1)

viene sostituito con l'uscita di

FCNJ(1, 1)

che vale

1

Analogamente:

(1→0)

viene sostituito con l'uscita di

FCND(1, 0)

che vale

0

5.3 FORMULE ED ENUNCIATI

Finora non abbiamo trattato in dettaglio la relazione fra gli enunciati e le formule ibride. Come si ricorderà, una stringa è una qualunque sequenza di caratteri. Abbiamo definito enunciato qualunque stringa che soddisfi le regole dalla 1 alla 9 dei Capitoli 3 e 4:

1. Una stringa costituita da una delle lettere singole 'A', 'B', ..., 'O' (eventualmente numerate) è un enunciato.
2. Se **P** è un enunciato,
   allora ¬**P** è un enunciato.
3. Se **P** e **Q** sono enunciati,
   allora **(P∧Q)** è un enunciato.
4. Se **P** e **Q** sono enunciati,
   allora **(P∨Q)** è un enunciato.
5. Se **P** e **Q** sono enunciati,
   allora **(P→Q)** è un enunciato.
6. Se **P** e **Q** sono enunciati,
   allora **(P↔Q)** è un enunciato.
7. Se **P** e **Q** sono enunciati,
   allora (**P** XOR **Q**) è un enunciato.
8. Se **P** e **Q** sono enunciati,
   allora (**P** NOR **Q**) è un enunciato.
9. Se **P** e **Q** sono enunciati,
   allora (**P** NAND **Q**) è un enunciato.
10. Solo le stringhe costruite secondo le regole dalla 1 alla 9 sono enunciati.

La regola 10 è detta *clausola di chiusura*. Per semplicità, nel seguito ometteremo di solito gli ultimi connettivi e non riporteremo esplicitamente la clausola di chiusura.

Definiremo *formula ibrida* una stringa ottenuta da un enunciato sostituendo tutte le lettere proposizionali con i loro valori di verità.

Come si determina se una formula non è stata generata in questo modo e quindi non deriva da un enunciato? Una stringa che sia un enunciato sarà detta *ben formata*.

Una risposta adeguata si può ricavare dall'algoritmo appena visto per il calcolo dei valori di verità degli enunciati molecolari. Prima di considerare questo ap-

proccio, tuttavia, conviene esaminare alcuni motivi per cui una stringa non può essere un enunciato. Vediamo alcuni esempi di stringhe che non sono enunciati e commentiamoli:

$(((A \wedge B) \vee C)$	Troppe parentesi aperte.
$((A \rightarrow B)) \wedge C)$	Troppe parentesi chiuse.
$((AB) \wedge C)$	Manca un connettivo fra ‘A’ e ‘B’.
$)A \wedge B($	Scambio fra parentesi aperta e chiusa.
$(A \rightarrow \rightarrow B)$	Due connettivi consecutivi.
$(A \neg \vee B)$	Due connettivi consecutivi.
$A \wedge$	Il connettivo $\wedge$ richiede due enunciati.

UN ALGORITMO PER DETERMINARE SE UNA FORMULA È BEN FORMATA

Con un po’ di esperienza si riesce ad individuare a colpo d’occhio una stringa non ben formata. Ma come si può essere davvero certi che la stringa in esame sia ben formata? Per rispondere a questa domanda occorre costruire un algoritmo che, applicato ad una stringa sospetta, emetta, correttamente e dopo un tempo finito, un messaggio come “No, non è ben formata” oppure “Sì, è ben formata”. Fortunatamente, buona parte del lavoro è già stata fatta e basta inserire alcuni passi nell’algoritmo CALCOLO DEL VALORE DI VERITÀ.
Gli errori nelle parentesi sono facilmente identificabili. Ricordando il funzionamento del primo contatore del sottoprogramma per l’individuazione delle sottoformule più interne [passo 5(d)], si può osservare che:

Se, dopo l’ultimo carattere, il contatore è maggiore di 0,
ci sono troppe parentesi aperte.
Se, dopo l’ultimo carattere, il contatore è minore di 0,
ci sono troppe parentesi chiuse.
Se, durante la scansione da sinistra a destra, il contatore assume valori negativi, le parentesi non si corrispondono in modo corretto.

All’algoritmo aggiungiamo un passo che, se si verifica una di queste condizioni, emette il messaggio “Non ben formata” e arresta immediatamente l’esecuzione. Come si trattano, invece, gli errori che non coinvolgono le parentesi? Per rispondere a questa domanda si deve esaminare ancor più in dettaglio il passo 5(e). Durante l’esame delle sottoformule più interne entro la formula via via ridotta, si era detto che ogni sottoformula più interna deve avere la struttura

(<valore di verità 1> ☆ <valore di verità 2>)

ove <valore di verità 1> è 0 o 1, <valore di verità 2> è 0 o 1 e ☆ è $\wedge$, $\vee$, $\leftrightarrow$,

oppure →; ad esempio: '(1∧0)' oppure '(0→1)'. Se almeno una sottoformula più interna non ha questa struttura, la stringa originaria non è ben formata. Inoltre, se le sottoformule più interne presentano sempre questa struttura, finché si giunge all'ultimo, singolo valore di verità, la stringa originaria è ben formata.
In altre parole, ad ogni stadio dell'analisi delle sottoformule più interne, la sottoformula in esame deve godere delle seguenti proprietà:

1. Contiene esattamente cinque simboli.
2. Il primo simbolo è '(',
   il secondo simbolo è un valore di verità ('0' o '1'),
   il terzo simbolo è un connettivo binario (∧, ∨, → o ↔),
   il quarto simbolo è un valore di verità ('0' o '1') e
   il quinto simbolo è ')'.

Nel nostro pseudolinguaggio di programmazione, un sottoprogramma, nel determinare se una sottoformula più interna soddisfa queste due proprietà, è:

SOTTOPROGRAMMA VERIFICA DI SOTTOFORMULA

FOR ogni sottoformula più interna del passo 5(d),

(a) IF la sottoformula non contiene esattamente cinque simboli,
 THEN OUTPUT "Non ben formata" e STOP.
(b) IF il primo simbolo non è '(',
 THEN OUTPUT "Non ben formata" e STOP.
(c) IF il secondo simbolo non è '0' o '1',
 THEN OUTPUT "Non ben formata" e STOP.
(d) IF il terzo simbolo non è un connettivo binario,
 THEN OUTPUT "Non ben formata" e STOP.
(e) IF il quarto simbolo non è '0' o '1',
 THEN OUTPUT "Non ben formata" e STOP.
(f) IF il quinto simbolo non è ')',
 THEN OUTPUT "Non ben formata" e STOP.

Questo sottoprogramma va inserito fra il passo 5(d) e il passo 5(e) dell'algoritmo CALCOLO DEL VALORE DI VERITÀ.
Il motivo dell'inserimento di questo sottoprogramma è in realtà molto semplice. Esso controlla ogni sottoformula identificata al passo 5(d) per verificare se ha la struttura corretta, poi ripete questa verifica man mano che la formula viene ridotta dal procedimento ciclico di sostituzione. Questo sottoprogramma, tuttavia, individua solo alcune delle caratteristiche che possono rendere non ben formata una stringa. Per permettere a CALCOLO DEI VALORI DI VERITÀ di identificare tutti gli errori, bisogna apportarvi modifiche più consistenti. Chiameremo VERIFICA DI ENUNCIATI l'algoritmo così ottenuto.

ALGORITMO VERIFICA DI ENUNCIATI

1. INPUT una stringa S.
2. IF S contiene simboli diversi dalle lettere proposizionali o dai connettivi.
   THEN OUTPUT "Non ben formata" e STOP.
3. Sostituire tutte le lettere proposizionali della stringa con valori di verità arbitrari (ad esempio, tutti '0' o tutti '1').
4. Scandendo la stringa da sinistra a destra, eliminare tutti i doppi simboli di negazione' ¬ ¬ '.
5. WHILE la stringa ha più di un carattere,
   (a) Scandendo la stringa da sinistra a destra, sostituire tutti i '¬ 1' con '0'.
   (b) Scandendo la stringa da sinistra a destra, sostituire tutti i '¬ 0' con '1'.
   (c) IF rimane solo un valore di verità
       THEN OUTPUT "Ben formata" e STOP.
   (d) IF non ci sono sottoformule più interne
       THEN OUTPUT "Non ben formata" e STOP.
   (e) FOR ogni sottoformula più interna:
       (i) VERIFICA DI SOTTOFORMULA.
       (ii) Sostituire la sottoformula con '0'.
6. OUTPUT "Ben formata".
7. STOP.

Si noti che, per applicare VERIFICA DI ENUNCIATI, non occorre fornire in ingresso i valori di verità degli enunciati atomici. Al passo 3, qualsiasi assegnamento arbitrario (ad esempio, 0 per ogni enunciato atomico) può andar bene per gli scopi di questo algoritmo. Ciò avviene in quanto VERIFICA DI ENUNCIATI controlla soltanto se la stringa d'ingresso è ben formata, ma non ha lo scopo di calcolarne il valore di verità (ammesso che si tratti effettivamente di un enunciato).

5.4 CONCLUSIONI

In questo capitolo abbiamo definito degli algoritmi per calcolare il valore di verità di un enunciato molecolare a partire dai valori di verità delle sue parti atomiche. In modo informale, si può dire che questo metodo consiste nel sostituire l'enunciato con una formula avente i valori di verità al posto degli enunciati atomici, e nell'applicare poi le funzioni di verità alle *sottoformule*, riducendo ciascuna di esse ad un singolo valore di verità. Ripetendo questo procedimento finché non rimangono più sottoformule, si giunge infine al valore di verità dell'enunciato originario. Per definire in modo preciso questo procedimento è fondamentale il concetto di *sottoformule più interne* (sottoformule circondate dal maggior numero

di parentesi).
L'algoritmo principale è CALCOLO DEL VALORE DI VERITÀ. Di esso abbiamo raffinato due sottoprogrammi importanti: il primo individua una sottoformula più interna in una stringa, contando le parentesi; l'altro sostituisce una sottoformula più interna col suo valore di verità, per mezzo delle funzioni di verità dei Capitoli 2 e 3.
Abbiamo fornito delle regole per definire un enunciato come una stringa ben formata: deve essere una lettera singola ('A', ..., 'O') oppure avere la forma $\neg\mathbf{P}$ o la forma $(\mathbf{P} \star \mathbf{Q})$, ove **P** e **Q** sono a loro volta enunciati e ☆ è un qualunque connettivo binario. Abbiamo anche definito degli algoritmi per controllare una stringa in base a queste regole.

5.5 ESERCIZI

A. Si consideri il seguente problema (che è una versione semplificata del progetto esposto in questo capitolo).
È data una formula che deriva o da una congiunzione o da una disgiunzione; ciascuno dei congiunti (o dei disgiunti) ha valore di verità 0 o 1. Esempi di formule di questo tipo possono essere:

$(1\wedge 0)$
$(0\vee 1)$
$(0\wedge 1)$

È dunque noto a priori che la stringa assegnata è costituita da cinque simboli e che il primo simbolo è '(', il secondo è un valore di verità (0 o 1), il terzo è un connettivo ($\wedge$ o $\vee$), il quarto è un valore di verità (0 o 1) e il quinto è ')'. Scrivere un algoritmo per determinare il valore di verità dell'intera espressione.

$(1\wedge 0)$

L'uscita dev'essere

0

Suggerimento: una delle prime istruzioni deve ricevere in ingresso (INPUT) la formula. L'algoritmo deve poi ben presto decidere se la formula ricevuta in ingresso deriva da una congiunzione o da una disgiunzione (cioè se il simbolo intermedio è $\wedge$ o $\vee$), dopodiché deve ramificarsi di conseguenza.
Anziché usare le funzioni FCNJ e FDSJ, si scriva una parte di algoritmo che svolga lo stesso compito.

B. Si consideri il primo contatore descritto nel raffinamento del passo 5(d) di

CALCOLO DEL VALORE DI VERITÀ.

1. Nel caso della stringa

   (A→(B∨ (¬C∧D)))

   quanto vale il primo contatore nelle posizioni seguenti?

   (a) 2
   (b) 4
   (c) 5
   (d) 7
   (e) 10
   (f) 11
   (g) 12

2. Nel caso della stringa

   (((A∧¬B)→¬C) ∨(C∧(D∧¬E)))

   quanto vale il primo contatore nelle posizioni seguenti?

   (a) 3
   (b) 5
   (c) 7
   (d) 9
   (e) 12
   (f) 15
   (g) 23
   (h) 24

C. Alle stringhe seguenti si applichi il secondo contatore, descritto nel raffinamento del passo 5(d).

Esempio: (A ∧ (B∧C))

Risposta: 4, 8

perché la sottoformula più interna inizia nella posizione 4 e finisce nella posizione 8.

1. (((A→B)∧C) ∨¬D)
2. ((A∧B)∧C)
3. ((A∧B)∧(C∧D))
4. ((A∧(B∨¬C)) →(C∧(D∨E)))
5. (¬(A∨¬C)→(F→(F∧F)))
6. ((A→ ¬(B∧C))→D)

D. Il passo 3 di CALCOLO DEL VALORE DI VERITÀ ("Sostituire ogni enunciato atomico con il valore di verità ad esso assegnato") non è stato dettagliato. Si consideri il caso semplificato di stringhe contenenti solo due lettere proposizionali distinte, 'A' e 'B'.

Scrivere un algoritmo che riceva in ingresso una stringa e sostituisca ognuna delle due lettere proposizionali con il valore di verità associato [V(A) o V(B)].

E. Alcune delle formule seguenti non derivano da stringhe ben formate. Si applichi VERIFICA DI ENUNCIATI, arrestandosi quando si scopre un errore. Si spieghi altresì qual è l'errore riscontrato.

1. (1¬0)
2. (1∧ (0∨∧1))

3. $((0\wedge 1)(1\rightarrow 1))$
4. $\neg(\neg 0\wedge 1)$
5. $((0\rightarrow 1)\vee(0(0\wedge 1)))$
6. $(((0\vee 1) \wedge 1)$
7. $\vee (0)$
8. $(\wedge(1\vee 0))$
9. $(1\rightarrow(0\vee(1\wedge 1)))$
10. $(((1\wedge\neg 0)\vee\neg(0\wedge 0)\rightarrow 1)$

F. Usando CALCOLO DEL VALORE DI VERITÀ e i valori di verità sotto indicati, determinare il valore di verità di ciascuno dei seguenti enunciati. Scrivere i passaggi e il risultato finale nel modo indicato al paragrafo 5.1.

1. Usando i valori di verità

   $V(A) = 0 \qquad V(C) = 1$

   $V(B) = 1 \qquad V(D) = 1$

   determinare i valori di verità degli enunciati da (a) a (r) sotto riportati.
2. Usando i valori di verità

   $V(A) = 1 \qquad V(C) = 0$

   $V(B) = 0 \qquad V(D) = 1$

   determinare i valori di verità degli enunciati da (a) a (r) sotto riportati:

   a. $\neg(A\rightarrow(B\wedge(C\vee D)))$
   b. $(C\wedge\neg(B\wedge D))$
   c. $(\neg\neg D\wedge(A\rightarrow(B\vee C)))$
   d. $(B\vee(C\vee\neg D))$
   e. $(A\wedge(B\wedge C))$
   f. $((A\wedge B)\wedge C)$
   g. $(A\wedge(B\vee C))$
   h. $((A\wedge B)\vee C)$
   i. $((A\wedge(B\rightarrow A))\rightarrow B)$
   j. $((A\wedge(A\rightarrow B))\rightarrow B)$
   k. $(B\rightarrow A)$
   l. $(\neg B\rightarrow\neg A)$
   m. $(A\rightarrow B)$
   n. $(\neg A\vee B)$
   o. $(A\wedge\neg B)$
   p. $\neg(A\wedge B)$
   q. $(\neg A\vee\neg B)$
   r. $(\neg A\wedge\neg B)$

5.6 SUGGERIMENTI PER UNA REALIZZAZIONE SU CALCOLATORE

Qualunque programma per calcolatore che realizzi CALCOLO DEL VALORE DI VERITÀ all'inizio deve ricevere in ingresso una stringa, che costituisce l'enunciato da esaminare. Il programma deve poi ricevere in ingresso, per ogni lettera proposizionale di tale stringa, il corrispondente valore di verità, espresso come TRUE o FALSE oppure come 0 o 1. Come vedremo, sarà importante che ogni simbolo significativo occupi una sola posizione; inoltre, si è vincolati all'uso dei simboli disponibili sulla tastiera; perciò converrà rappresentare il condizionale ($\rightarrow$) con un altro simbolo, ad esempio $>$. La stessa sostituzione va fatta per $\neg$, $\wedge$, $\vee$ o per altri simboli non disponibili. Se è possibile scegliere, spesso risulterà opportuno utilizzare solo lettere maiuscole per rappresentare gli enunciati atomici.
La realizzazione dell'algoritmo CALCOLO DEL VALORE DI VERITÀ mediante un programma per calcolatore può richiedere l'uso frequente di funzioni per la manipolazione delle stringhe. La disponibilità di tale tipo di funzioni varia notevolmente da linguaggio a linguaggio. Di conseguenza, si possono fornire ben poche indicazioni specifiche per la stesura di un effettivo programma.
Si consideri tuttavia il passo 1 di CALCOLO DEL VALORE DI VERITÀ. Sia STRINGA il nome della stringa d'ingresso. Si supponga di aver già identificato gli enunciati atomici di STRINGA e di aver deciso di memorizzarli in un vettore (cioè in una successione finita di variabili) di nome ATOMICO(n); ad esempio, ATOMICO(1) = 'A', ATOMICO(2) = 'B', eccetera. Il valore di verità di ciascuno di questi enunciati atomici può essere memorizzato in un vettore corrispondente, che chiameremo VALORE(n); ad esempio, VALORE(1) = 1, VALORE(2) = 0, eccetera. Naturalmente ATOMICO(n) e VALORE(n) hanno lo stesso numero di elementi. Se si lavora in un linguaggio come il Pascal o il PL/I, tali vettori corrispondenti possono essere realizzati mediante un unico "record" con due campi.
Due funzioni per l'elaborazione delle stringhe sono particolarmente utili:

LENGTH(STRINGA)

che calcola la lunghezza della stringa STRINGA, ossia il numero dei simboli in essa contenuti, e

MID(STRINGA, n)

che seleziona il simbolo n-esimo della stringa STRINGA. Ad esempio, se STRINGA = 'LOGICA', si ha

LENGTH(STRINGA) = 6

perché

LENGTH(‘LOGICA’) = 6

e

MID(STRINGA, 3) = ‘G’

perché il terzo simbolo di ‘LOGICA’ è ‘G’.
Vediamo dunque un raffinamento del passo 1 che usa questi vettori e queste funzioni; in esso, l’istruzione

```
FOR<variabile> = <valore 1> TO <valore 2>
```

significa “per <variabile> che va da <valore 1> a <valore 2>”.

```
1.1       INPUT STRINGA.
1.2       LET k = numero degli enunciati atomici di STRINGA.
1.3       FOR n = 1 TO LENGTH(STRINGA)
          (a)  FOR m = 1 TO k
               (i)  IF MID(STRINGA, n) = ATOMICO(m)
                         THEN OUTPUT VALORE(m).
               (ii) IF MID(STRINGA, n) = (
                         oppure MID(STRINGA, n) = )
                         oppure MID(STRINGA, n) = ∧
                         oppure MID(STRINGA, n) = ∨
                         oppure MID(STRINGA, n) = >
                         THEN OUTPUT MID(STRINGA, n).
```

Le due istruzioni di OUTPUT dei passi 1.3(a)(i) e 1.3(a)(ii) devono essere usate per creare una nuova stringa, che chiameremo NUOVASTRINGA, la quale è uguale a STRINGA, salvo per il fatto che contiene i valori di verità al posto delle lettere proposizionali. Se indichiamo con ‘ + ’ l’operazione di *concatenazione*, che consiste nell’unire più stringhe a formarne una sola (ad esempio, ‘LO’ + ‘GICA’ = ‘LOGICA’), tali passi di OUTPUT possono essere raffinati nel modo seguente (supponendo che a NUOVASTRINGA venga assegnato come valore iniziale lo spazio bianco ‘ ’):

```
1.3(a)(i)  IF MID(STRINGA, n) = ATOMICO(m)
              THEN LET NUOVASTRINGA = NUOVASTRINGA +
              VALORE(m).
1.3(a)(ii) IF MID(STRINGA, n) = (
                oppure MID(STRINGA, n) = )
                oppure MID(STRINGA, n) = ∧
                oppure MID(STRINGA, n) = ∨
                oppure MID(STRINGA, n) = >
               THEN LET NUOVASTRINGA = NUOVASTRINGA +
               MID(STRINGA, n).
```

Benché questo abbozzo di programma possa a prima vista apparire molto complicato, in realtà esso descrive un procedimento per nulla difficile. Il ciclo FOR esterno, al passo 1.3, scandisce STRINGA simbolo per simbolo. Il ciclo FOR interno, al passo 1.3(a), controlla ogni lettera proposizionale rappresentante un enunciato atomico ('A', 'B', ...). Il test al passo 1.3(a)(i) verifica se il simbolo di STRINGA attualmente in esame è una lettera proposizionale. Se lo è, il suo valore di verità viene aggiunto a NUOVASTRINGA per concatenazione. Se non lo è (passo 1.3(a)(ii), il simbolo viene direttamente copiato in NUOVASTRINGA, a meno che non sia uno spazio bianco o un altro simbolo non significativo. Il procedimento ha globalmente l'effetto di trasformare STRINGA in NUOVASTRINGA, copiando i simboli in NUOVASTRINGA, eccetto le lettere proposizionali, delle quali in NUOVASTRINGA vengono copiati i valori di verità.
Il primo contatore, descritto nel raffinamento del passo 5(d) di CALCOLO DEL VALORE DI VERITÀ, può essere realizzato nel modo seguente. Impieghiamo ancora le funzioni LENGTH e MID precedentemente definite, ma usiamo > e + con il loro normale significato aritmetico:

1. LET m = 0.
2. FOR n = 1 TO LENGTH (NUOVASTRINGA)
   (a) IF MID(NUOVASTRINGA, n) = (
       THEN LET m = m + 1.
   (b) IF MID(NUOVASTRINGA, n) =)
       THEN LET m = m — 1.
   (c) IF m >r
       THEN LET r = m.

Il ciclo FOR scandisce NUOVASTRINGA da sinistra a destra e il passo 2(c) memorizza il massimo valore di m nella variabile r.
Se si vuole che il programma determini anche se le parentesi sono usate correttamente, si possono aggiungere i passi seguenti:

2(b)(ii) IF $m<0$
    THEN OUTPUT "Errore nell'uso delle parentesi."
3(a). IF $m>0$
    THEN OUTPUT "Troppe parentesi aperte."
3(b). IF $m<0$
    THEN OUTPUT "Troppe parentesi chiuse."

6

Logica enunciativa: algoritmi per generare tavole di verità e per determinare la validità di un'argomentazione

Nei Capitoli 3 e 4 abbiamo spesso usato tavole di verità, come descrizioni estensionali di funzioni di verità. Si tratta di rappresentazioni grafiche bidimensionali che illustrano la relazione funzionale fra il valore di verità di un enunciato molecolare e i valori di verità delle sue parti. In matematica una tavola di questo tipo è detta *matrice*; nel gergo informatico viene detta *array bidimensionale.*
Per la disgiunzione avevamo scritto:

V(**P**)	V(**Q**)	V(**P**∨**Q**)
0	0	0
0	1	1
1	0	1
1	1	1

Le due colonne di sinistra illustrano le possibili combinazioni di verità (1) e di falsità (0) che possono essere assunte dagli enunciati **P** e **Q** in diverse situazioni. Possiamo considerare ogni riga come la descrizione di una situazione. Finora la colonna più a destra conteneva i valori di verità di un solo enunciato molecolare, in corrispondenza delle situazioni descritte dai valori di verità degli enunciati che ne costituiscono le parti verofunzionali. Ad esempio, nella tavola sopra riportata, la prima riga descrive la situazione in cui sia **P** che **Q** hanno valore di verità FALSE; in tale situazione, anche la disgiunzione (**P**∨**Q**) ha valore di verità FALSE. La seconda riga descrive una situazione in cui la disgiunzione (**P**∨**Q**) ha valore di verità TRUE; e così via per le altre righe.
Nel Capitolo 5 abbiamo descritto per esteso il calcolo del valore di verità di un enunciato molecolare comunque complesso, a partire dai valori di verità delle sue parti atomiche. Siamo ora in grado di generare tavole di verità più complesse, come:

V(**P**)	V(**Q**)	V(**P**→(**P**→(**P**∧**Q**)))
0	0	1
0	1	1
1	0	0
1	1	1

I valori riportati nell'ultima colonna si calcolano mediante l'algoritmo del Capitolo 5. Per la prima riga, si sostituisce ogni occorrenza di **P** con '0' e ogni occorrenza di **Q** con '1', poi si calcola il valore della formula ibrida così ottenuta:

$$(0\rightarrow(0\rightarrow(0\wedge 0)))$$

valore che risulta pari a 1. Si sarebbe potuto giungere a questa conclusione anche osservando che $V(0\rightarrow n) = 1$ qualunque sia il valore di n.
Una tavola di verità può anche essere usata per illustrare il modo in cui *più* enunciati molecolari dipendono dai valori di verità delle loro parti atomiche. Come vedremo tra breve, questo tipo di tavola di verità risulta molto utile per la determinazione della validità o non validità delle argomentazioni nella logica enunciativa. Si consideri la seguente tavola di verità:

V(**P**)	V(**Q**)	V(**R**)	V(**P**→**Q**)	V(**Q**→**R**)	V(**P**→**R**)
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	1	1

La prima riga mostra che, quando **P** vale FALSE e **Q** vale FALSE e **R** vale FALSE, il valore di verità di (**P**→**Q**) è TRUE, il valore di verità di (**Q**→**R**) è TRUE e il valore di verità di (**P**→**R**) è TRUE. Nella tavola di verità sono riportati i valori di verità di (**P**→**Q**), (**Q**→**R**) e (**P**→**R**) in funzione di tutte le combinazioni dei valori di verità delle loro parti atomiche.

6.1 TAVOLE DI VERITÀ GENERALIZZATE

La tavola di verità precedente suggerisce la seguente generalizzazione del formato di una tavola di verità.

Valori di verità di enunciati atomici	**Valori di verità di enunciati dipendenti da tali enunciati atomici**
Situazioni: tutte le possibili combinazioni di valori di verità degli enunciati atomici	

Si possono fare alcune osservazioni di carattere generale sulle tavole di verità:

1. Gli enunciati atomici da riportare nelle colonne di sinistra sono quelli contenuti negli enunciati componenti. Ad esempio, se si stanno considerando gli enunciati **(P→Q)**, **(Q→R)** e **(R→S)**, gli enunciati atomici da riportare sono **P**, **Q**, **R** e **S**.
2. Se si stanno considerando *n* enunciati atomici, le possibili combinazioni dei loro valori di verità sono 2***n* (la notazione '*a****n*' significa '*a* elevato alla *n*-esima potenza').
   Ogni enunciato atomico, infatti, può assumere solo *due* valori di verità. Per due enunciati atomici esistono allora 2**2 = 2 × 2 = 4 combinazioni (00, 01, 10, 11); per tre enunciati atomici ne esistono 2**3 = 2 × 2 × 2 = 8, e così via.
3. Solitamente, per disporre sulle righe i valori di verità degli enunciati atomici in modo da coprire tutte le 2***n* combinazioni, si usa il metodo seguente. Nella riga 1 si pone la rappresentazione binaria del numero 0, allineata a destra. Nella riga 2 si pone la rappresentazione binaria del numero 1, sempre allineata a destra. In generale, nella riga *k* si pone la rappresentazione binaria del numero *k* − 1, allineata a destra. L'ultima riga è una stringa costituita da *n* cifre '1'. Infine, gli spazi bianchi rimasti sotto gli enunciati atomici vengono riempiti con zeri.

Si supponga, ad esempio, di avere tre enunciati atomici, **P**, **Q** e **R**. La tavola di verità deve avere allora 2**3 = 8 righe. La prima riga contiene uno '0', spostato all'estrema destra; la seconda riga contiene un '1', la terza '10', la quarta '11', e così via. Si ottiene così il seguente risultato:

V(**P**)	V(**Q**)	V(**R**)	
		0	
		1	
	1	0	
	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Si completa il procedimento riempiendo gli spazi bianchi con degli zeri. Si ottiene così:

V(**P**)	V(**Q**)	V(**R**)	
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Si completa la tavola di verità riportando, nelle intestazioni delle colonne di destra, gli enunciati di cui interessano i valori di verità. Nelle righe sottostanti si scrivono poi tali valori di verità, calcolati applicando CALCOLO DEL VALORE DI VERITÀ ai valori di verità degli enunciati atomici riportati nelle corrispondenti righe della parte sinistra della tavola.

6.2 L'ALGORITMO GENERATORE DI TAVOLE DI VERITÀ

Dopo aver descritto in modo informale la costruzione delle tavole di verità, possiamo ora fornire un algoritmo che le generi.

ALGORITMO GENERATORE DI TAVOLE DI VERITÀ

1. INPUT gli enunciati di cui si vogliono studiare le relazioni verofunzionali.
2. LET *m* = numero di tali enunciati (Per un'argomentazione, *m* è il numero di premesse più uno (la conclusione).)

3. LET n = numero degli enunciati atomici distinti che compaiono in tali enunciati.
4. Creare una tavola con $n + m$ colonne e $2{*}{*}n$ righe.
5. FOR ogni enunciato atomico **S**
   (a) Scrivere 'V(**S**)' in testa alle colonne, cominciando da sinistra.
6. FOR ogni enunciato **S** di cui si vogliono studiare le relazioni verofunzionali
   (a) Scrivere 'V(**S**)' in testa alle colonne rimanenti, cominciando dalla colonna $n + 1$.
7. (a) Contare in binario da 0 a $(2{*}{*}n - 1)$.
   (b) Scrivere questi numeri binari uno per riga, allineati a destra sotto le prime n colonne, con una cifra per colonna (in tal modo, nella prima riga la n-esima colonna conterrà uno '0'; nella seconda riga, la n-esima colonna conterrà un '1'; nella terza riga, la $(n - 1)$-esima colonna conterrà un '1' e la n-esima colonna conterrà uno '0', e così via).
   (c) Dopo aver compilato l'ultima riga, tornare alle righe precedenti, mettendo uno '0' in tutti gli spazi vuoti rimasti nelle prime n colonne.
8. FOR ogni colonna **c** FROM $n + 1$ TO $n + m$
   (a) FOR ogni riga **r** FROM 1 TO $2{*}{*}n$
      i. OUTPUT il valore di verità ottenuto applicando CALCOLO DEL VALORE DI VERITÀ all'enunciato della colonna **c** e ai valori di verità della riga **r**.
9. STOP.

Ad esempio, se i tre enunciati iniziali sono (**P**→**Q**), (**Q**→**R**) e (**P**→**R**), questo algoritmo fornisce in uscita la seguente tavola di verità:

V(**P**)	V(**Q**)	V(**R**)	V(**P**→**Q**)	V(**Q**→**R**)	V(**P**→**R**)
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	1	1

6.3 DETERMINAZIONE DELLA VALIDITÀ DI UN'ARGOMENTAZIONE

In logica enunciativa, le tavole di verità possono essere usate per determinare se un'argomentazione è valida o no. Per rendersene conto, conviene richiamare le definizioni di argomentazione e di argomentazione valida. Un'argomentazione è un insieme di enunciati, uno dei quali, detto conclusione, è posto come conseguenza degli altri, detti premesse. Un'argomentazione è valida se e solo se

> la conclusione ha valore di verità TRUE in ogni situazione in cui le premesse hanno tutte valore di verità TRUE

o, equivalentemente,

> non c'è nessuna situazione in cui le premesse abbiano tutte valore di verità TRUE e la conclusione abbia valore di verità FALSE.

Un'argomentazione non è valida se e solo se

> esiste qualche situazione in cui le premesse hanno tutte valore di verità TRUE e la conclusione ha valore di verità FALSE.

In logica enunciativa, le "situazioni" sono semplicemente diverse combinazioni di verità e falsità di enunciati *atomici* (descritte dalle prime n colonne). Le tavole di verità sono state costruite in modo da comprendere tutte le situazioni possibili: ogni riga rappresenta una situazione diversa, e le diverse righe, nel loro complesso, coprono tutte le situazioni possibili.
Una tavola di verità può essere usata per decidere se un'argomentazione è valida oppure no: si utilizzano per le premesse le colonne di destra, esclusa l'ultima, cioè le colonne dalla $(n + 1)$-esima alla $(n + m - 1)$-esima, mentre l'ultima colonna, cioè la $(n + m)$-esima, viene utilizzata per la conclusione. Si ragiona poi nel modo seguente:

> Se c'è almeno una riga che contenga un '1' (TRUE) in ogni colonna-premessa e uno '0' (FALSE) nella colonna-conclusione, allora l'argomentazione non è valida.
> Se non ci sono righe cosiffatte, allora l'argomentazione è valida.

Si consideri l'argomentazione seguente:

1. Se l'Unione Sovietica respinge il trattato proposto, allora l'accordo finale sarà rinviato.
2. L'Unione Sovietica respinge il trattato proposto.
   Perciò
3. L'accordo finale sarà rinviato.

Questa argomentazione potrebbe essere rappresentata simbolicamente nel modo seguente:

1. (A→B)
2. A

∴ 3. B

Il simbolo ∴ rappresenta gli indicatori convenzionali di conclusione 'perciò' e 'dunque'.

L'argomentazione è valida? Usando una tavola di verità nel modo appena proposto, si può rispondere a tale domanda nella seguente maniera. Si costruisce una tavola di verità, usando alcuni dei procedimenti descritti nell'algoritmo GENERATORE DI TAVOLE DI VERITÀ.

Passo 1 - Nell'argomentazione ci sono tre enunciati: due premesse e una conclusione. Perciò $m = 3$.
Passo 2 - Questi enunciati contengono solo due enunciati atomici: A e B. Perciò $n = 2$.
Passo 3 - La tavola di verità ha dunque $n + m = 2 + 3 = 5$ e $2{*}{*}n = 2{*}{*}2 = 4$ righe.
Passi 4-7 - Scrivendo le intestazioni delle colonne e compilando le righe si ottiene:

V(A)	V(B)	V(A→B)	V(A)	V(B)
0	0	1	0	0
0	1	1	0	1
1	0	0	1	0
1	1	1	1	1

Questa rappresentazione contiene molte informazioni. La prima riga esprime il fatto che, nella situazione in cui gli enunciati 'L'Unione Sovietica respinge il trattato proposto' e 'L'accordo finale sarà rinviato' hanno entrambi valore di verità FALSE, l'enunciato 'Se l'Unione Sovietica respinge il trattato proposto, allora l'accordo finale sarà rinviato' ha valore di verità TRUE. In questa situazione, la prima premessa vale TRUE, la seconda premessa vale FALSE e la conclusione vale FALSE. Ognuna delle ultime tre righe descrive una situazione diversa e indica se le premesse e la conclusione hanno valore di verità TRUE oppure FALSE in tale situazione.

Si possono usare queste informazioni per determinare se l'argomentazione è valida o non valida. Se esiste una riga in cui entrambe le premesse abbiano valore di verità TRUE e la conclusione abbia valore di verità FALSE, allora l'argomentazione non è valida; se non esistono righe cosiffatte, allora l'argomentazione è vali-

da. Consultando la tavola di verità sopra riportata si vede che non esistono righe in cui le premesse valgano TRUE e la conclusione valga FALSE. Perciò l'argomentazione è valida.

In effetti, non solo è valida *questa* argomentazione, bensì *ogni* argomentazione della stessa *forma* risulta ugualmente valida. In altre parole, ogni argomentazione avente la forma

Se <enunciato 1> allora <enunciato 2>
<enunciato 1>
∴ <enunciato 2>

è valida. Usando la tradizionale terminologia latina medievale, si dice che le argomentazioni aventi questa forma usano lo schema di inferenza *modus ponens*. Ogni applicazione del *modus ponens* è valida.

Consideriamo un'altra argomentazione:

1. Se Luca fa uso di eroina, allora Luca un tempo fumava marijuana.
2. Luca un tempo fumava marijuana.
   Perciò
3. Luca fa uso di eroina.

Espressa in forma simbolica, l'argomentazione diventa:

1. $(E \rightarrow M)$
2. M

∴ 3. E

Prima di continuare, conviene notare la differenza fra la struttura di questo esempio e la struttura dell'esempio precedente. In questo esempio, la seconda premessa è il *conseguente* del condizionale che costituisce la prima premessa, mentre la conclusione è l'*antecedente* di tale condizionale. La struttura di questa nuova argomentazione è dunque:

Se <enunciato 1> allora <enunciato 2>
<enunciato 2>
∴ <enunciato 1>

Le *fallacie* sono tipi di argomentazione non validi che purtroppo vengono spesso usati nella vita quotidiana. La fallacia rappresentata in quest'ultimo esempio è detta *affermazione del conseguente*. Nel primo esempio, la seconda premessa era l'*antecedente* del condizionale, mentre la conclusione ne era il *conseguente*.

La tavola di verità di questa nuova argomentazione è:

V(E)	V(M)	V(E→M)	V(M)	V(E)
0	0	1	0	0
0	1	1	1	0
1	0	0	0	1
1	1	1	1	1

In questo caso si vede che esiste una riga, la seconda, in cui entrambe le premesse hanno valore di verità TRUE, ma la conclusione ha valore di verità FALSE. La seconda riga rappresenta la situazione in cui Luca ha fumato marijuana ma non fa uso di eroina. In questa situazione, entrambe le premesse valgono TRUE, ma la conclusione vale FALSE. Poiché esiste una tale riga, l'argomentazione non è valida. In un'argomentazione valida, la verità delle premesse garantisce la verità della conclusione. Al contrario, come abbiamo visto, la verità delle premesse di questa argomentazione non garantisce la verità della conclusione.

6.4 DETERMINAZIONE DI VALIDITÀ/NON VALIDITÀ

Il metodo delle tavole di verità che abbiamo appena esaminato costituisce il fondamento di un algoritmo per determinare se un'argomentazione espressa simbolicamente è valida o meno nella logica enunciativa. Se correttamente applicato, esso fornirà sempre la risposta corretta in un tempo finito, e la sua applicazione non richiede né intuizione né immaginazione.
Descritto per esteso, il nostro metodo per valutare la validità di un'argomentazione in logica enunciativa è costituito dai passi seguenti:

ALGORITMO DETERMINAZIONE DI VALIDITÀ/NON VALIDITÀ

1. INPUT un'argomentazione espressa in forma simbolica.
2. Applicare GENERATORE DI TAVOLE DI VERITÀ.
   (*Nota*: al passo 6 di GENERATORE DI TAVOLE DI VERITÀ la colonna $(n + m)$-esima deve contenere i valori di verità della conclusione)
3. IF esiste una riga che contenga un '1' in corrispondenza di ogni colonna-premessa (cioè in corrispondenza delle colonne dalla $(n + 1)$-esima alla $(n + m - 1)$-esima) e uno '0' in corrispondenza dell'ultima colonna
   THEN (a) OUTPUT "Argomentazione non valida".
   (b) STOP.
4. IF non esiste una riga cosiffatta
   THEN (a) OUTPUT "Argomentazione valida".
   (b) STOP.

Il passo critico è ovviamente il passo 3, poiché è quello che determina effettivamente se l'argomentazione è valida o no. Lo si può raffinare nel modo seguente:

3.1. FOR ogni riga
    (a) LET COLONNA = $n + 1$.
    (b) WHILE COLONNA $\leqslant n + m - 1$ e compare un '1' in questa riga e in questa colonna
        LET COLONNA = COLONNA + 1
    (c) IF COLONNA = $n + m$
        THEN IF compare uno '0' in questa riga e in questa colonna
            THEN (i) OUTPUT "Argomentazione non valida".
                (ii) STOP.

Questa procedura comincia scandendo la prima riga (passo 3.1), a partire dalla prima colonna-premessa, la $n + 1$ (passo 3.1(a)). Finché si incontrano premesse vere, la scansione continua (passo 3.1(b)). Se si trova una premessa falsa, si scandisce la riga successiva, a partire dalla prima premessa. Se tutte le premesse hanno valore di verità TRUE, si esamina la conclusione, nella colonna $n + m$ (passo 3.1(c)). Se la conclusione ha valore di verità FALSE, l'argomentazione non è valida (passo 3.1(c)(i)) e l'algoritmo termina (passo 3.1(c)(ii)); altrimenti si scandisce la riga successiva, a partire dalla prima premessa. Se si sono esaminate tutte le righe senza trovarne nessuna che abbia tutte le premesse vere e la conclusione falsa, l'argomentazione è valida (passo 4).

6.5 METODI ALTERNATIVI

Se gli enunciati atomici contenuti nelle premesse e nella conclusione sono piuttosto numerosi, la tavola di verità che ne risulta può raggiungere dimensioni eccessive. Per la semplice argomentazione seguente (che non è valida)

1. (A→B)
2. (B→C)
3. (C→D)
4. (D→E)
5. E

∴ 6. A

la tavola di verità deve avere 2**5 = 32 righe e 11 colonne. Poiché la tavola di verità ha 32 × 11 caselle, occorre scrivervi 352 valori di verità. In altri casi si può aver a che fare con enunciati molecolari molto complessi, come

$$((A \rightarrow (\neg B \vee (C \wedge \neg D))) \vee \neg(\neg D \rightarrow A))$$

Il valore di verità di ogni enunciato di questo tipo deve essere ricalcolato ad ogni riga, usando CALCOLO DEL VALORE DI VERITÀ. In conclusione, il metodo delle tavole di verità può comportare un notevole impiego di carta, inchiostro e tempo prezioso.
Se si programma un calcolatore per applicare questo procedimento, CALCOLO DEL VALORE DI VERITÀ non presenta gravi difficoltà. Il calcolatore non usa carta e inchiostro per effettuare i suoi calcoli; la tavola di verità è infatti rappresentata mediante stati elettrici entro posizioni di memoria. Anche le argomentazioni contenenti molti enunciati atomici, oppure costituite da enunciati molecolari lunghi e complessi, richiedono in tal caso un impiego di tempo non nostro, bensì del calcolatore.
In linea di principio, dunque, potremmo usare un calcolatore per applicare il metodo di determinazione della validità basato sulle tavole di verità ad argomentazioni di dimensioni o complessità altrimenti intrattabili. Tuttavia, non sempre si può avere a disposizione un calcolatore, oppure si può non essere così abili o pazienti da riuscire a programmarlo correttamente. Inoltre il metodo completo delle tavole di verità, con il suo intrico di valori di verità, di rado assomiglia al modo in cui ragioniamo effettivamente. Non è verosimile che una persona, per ideare un'argomentazione, usi il metodo delle tavole di verità. Ed è ancor meno verosimile che le persone cui viene sottoposta un'argomentazione, anche se si tratta di studiosi di logica, la giudichino valida o non valida in base a una tavola di verità completa costruita a mente. Ci si può allora chiedere: esistono metodi più rapidi, più semplici o più naturali per determinare se un'argomentazione è valida oppure no?
In effetti esistono metodi con queste caratteristiche, ma non sono potenti come si potrebbe desiderare. In un cosiddetto *sistema formale di deduzione* si può costruire una *prova* di un'argomentazione valida. Se si riesce a trovare una prova corretta, si è certi che l'argomentazione è valida. Purtroppo, però, se non si riesce a trovare una prova, non si può concludere che l'argomentazione non è valida. La mancata individuazione di una prova può essere dovuta ad incapacità, o ad un impegno insufficiente nella ricerca della stessa. In tal caso, l'argomentazione potrebbe ugualmente essere valida; ci si potrebbe allora ricondurre al metodo delle tavole di verità per determinare se è valida o no. Nei Capitoli 8 e 9 tratteremo il metodo per dimostrare la validità di un'argomentazione fornendone una prova.

6.6 ALGORITMO DI WANG

Un altro algoritmo per determinare se un'argomentazione espressa in forma simbolica è valida oppure no è l'ALGORITMO DI WANG, dal nome del logico Hao Wang. L'ALGORITMO DI WANG, come l'algoritmo basato sulle tavole di verità DETERMINAZIONE DI VALIDITÀ/NON VALIDITÀ, permette sempre di determinare se un'argomentazione espressa in forma simbolica in logica enunciativa è valida oppure no. Al contrario però dell'algoritmo basato sulle tavole di

verità, non richiede la creazione di grandi tabelle.
L'idea che sta alla base dell'ALGORITMO DI WANG è molto semplice. Quando si costruisce una tavola di verità per determinare la validità di un'argomentazione, si cerca una situazione in cui tutte le premesse abbiano valore di verità TRUE, ma la conclusione abbia valore di verità FALSE. Se una tale situazione esiste, l'argomentazione non è valida; l'argomentazione è invece valida se una tale situazione non esiste. Ragionando in termini di tavole di verità, si cerca una *riga* in cui tutte le premesse valgano TRUE e la conclusione valga FALSE. L'ALGORITMO DI WANG è concepito per individuare questa situazione (riga) senza dover esaminare tutte le situazioni. L'algoritmo permette anche di determinare se una tale situazione non esiste.
Prima di descrivere dettagliatamente l'ALGORITMO DI WANG, consideriamo alcuni esempi che illustrino i principi su cui esso si fonda. Si consideri l'argomentazione seguente:

1. A
2. B
∴ 3. A

Consideriamo un tentativo di rendere vere le premesse e falsa la conclusione. A tal fine, scriviamo gli enunciati di questa argomentazione in modo diverso. A sinistra scriviamo la lista degli enunciati a cui tentiamo di attribuire il valore di verità TRUE, mentre a destra riportiamo la lista degli enunciati a cui tentiamo di attribuire il valore di verità FALSE. Separiamo poi le due colonne con una linea verticale. Nel caso di questa argomentazione scriviamo:

A	A
B	

Cerchiamo cioè di attribuire agli enunciati A e B il valore di verità TRUE e, nella stessa situazione, cerchiamo di attribuire all'enunciato A il valore di verità FALSE. È però evidente che ad uno stesso enunciato non si possono attribuire entrambi i valori di verità TRUE e FALSE nella stessa situazione. Poiché l'enunciato A compare sia nella lista degli enunciati cui cerchiamo di attribuire il valore di verità TRUE, sia nella lista degli enunciati cui cerchiamo di attribuire il valore di verità FALSE, è evidente che questo tentativo fallisce. In base a questo risultato si può fare una prima osservazione riguardo a queste liste:

> Se uno stesso enunciato compare sia nella lista di sinistra, sia nella lista di destra, non si riesce ad attribuire il valore di verità TRUE agli enunciati della lista di sinistra e FALSE a quelli della lista di destra. Si dice in tal caso che il tentativo *è fallito*.

Consideriamo un'altra argomentazione, più insolita:

1. C
2. ¬C
∴ 3. D

Usando il metodo delle liste di enunciati sinistra e destra, scriveremo ora:

C	D
¬ C	

indicando con ciò il fatto che cerchiamo di attribuire il valore di verità TRUE a C e ¬C e FALSE a D. Ma attribuire il valore di verità TRUE a ¬C equivale ad attribuire FALSE a C. La situazione in cui ¬C vale TRUE coincide con la situazione in cui C vale FALSE. Si possono quindi trasformare le liste spostando ¬C dalla lista di sinistra a quella di destra e rimuovendo nel contempo il segno di negazione. Si ottiene così il seguente risultato:

C	D
	C

Ci siamo così ricondotti al caso precedente: lo stesso enunciato compare in entrambe le liste. Abbiamo già visto che un tale tentativo fallisce. A proposito del trattamento del segno di negazione, possiamo fare una seconda osservazione:

> Se in una lista compare un enunciato negato, lo si sposta nell'altra lista, togliendogli il segno di negazione più esterno.

Si consideri ora questa argomentazione:

1. (A∧B)
∴ 2. B

Trasformandola nella coppia di liste, si ottiene

(A∧B) | B

In che modo si può attribuire il valore di verità TRUE ad '(A∧B)', cioè al primo enunciato della lista di sinistra? Poiché si tratta di una congiunzione, c'è un solo modo: entrambi i congiunti A e B devono valere TRUE. In altre parole, la coppia di liste sopra riportata può essere trasformata nel modo seguente:

A	B
B	

È evidente che anche questo tentativo fallisce, perché uno stesso enunciato compare in entrambe le liste. Il trattamento della congiunzione conduce ad una terza osservazione:

> Se una congiunzione compare nella lista di sinistra, si elimina tale congiunzione e si aggiungono separatamente alla lista di sinistra i due congiunti che la costituiscono.

Un'operazione analoga può essere effettuata su un enunciato della lista di destra:

> Se una disgiunzione compare nella lista di destra, si elimina tale disgiunzione e si aggiungono separatamente alla lista di destra i due disgiunti che la costituiscono.

La giustificazione di questa operazione è semplice: la lista di destra contiene gli enunciati cui si cerca di attribuire il valore di verità FALSE; ma una disgiunzione può essere resa falsa solo rendendo falsi entrambi i suoi disgiunti.
Abbiamo finora ricavato tre regole per la manipolazione delle coppie di liste:

1. Se in una lista compare un enunciato negato, lo si sposta nell'altra lista, togliendogli il segno di negazione più esterno.
2. Se nella lista di sinistra compare una congiunzione, la si sostituisce con i suoi due congiunti, elencati separatamente.
3. Se nella lista di destra compare una disgiunzione, la si sostituisce con i suoi due disgiunti, elencati separatamente.

Abbiamo anche introdotto una generalizzazione per emettere un giudizio definitivo su una coppia di liste:

> Se un enunciato compare in entrambe le liste, il tentativo è fallito.

Prima di continuare, applichiamo questi criteri a un'altra argomentazione:

  1. $\neg\neg(A\wedge(B\vee C))$
$\therefore$ 2. $(A\vee D)$

La prima coppia di liste che si può formare è:

$\neg\neg(A\wedge(B\vee C)) \mid (A\vee D)$

Spostiamo poi a destra la negazione '$\neg\neg(A\wedge(B\vee C))$' che compare nella lista di sinistra, avendo cura di eliminare solo il segno di negazione più esterno. Abbiamo così:

$$\begin{array}{l|l} & (A \vee D) \\ & \neg(A \wedge (B \vee C)) \end{array}$$

La lista di sinistra è momentaneamente vuota. L'enunciato '$\neg(A \wedge (B \vee C))$', che compare nella lista di destra, è però ancora una negazione. Applicando allora nuovamente la regola 1 otteniamo:

$$\begin{array}{l|l} (A \wedge (B \vee C)) & (A \vee D) \end{array}$$

L'enunciato che ora si trova nella lista di sinistra è una congiunzione. Si può quindi applicare la regola 2, che tratta le congiunzioni appartenenti alla lista di sinistra; si ottiene così:

$$\begin{array}{l|l} A & (A \vee D) \\ (B \vee C) & \end{array}$$

L'enunciato '$(A \vee D)$' nella lista di destra è una disgiunzione; in base alla regola 3 otteniamo perciò:

$$\begin{array}{l|l} A & A \\ (B \vee C) & D \end{array}$$

A questo punto si vede che l'enunciato A compare in entrambe le liste. In virtù della generalizzazione che ci permette di giudicare la coppia di liste, concludiamo che il tentativo di attribuire il valore di verità TRUE agli enunciati della lista di sinistra e FALSE a quelli della lista di destra è fallito. Poiché col metodo delle coppie di liste non siamo riusciti a rendere vere le premesse e falsa la conclusione, possiamo concludere che l'argomentazione considerata è valida, cioè che non esistono situazioni in cui tutte le premesse valgano TRUE e la conclusione valga FALSE.
Finora abbiamo esaminato solo argomentazioni valide. Si consideri l'argomentazione seguente:

1. $(A \wedge \neg B)$

$\therefore$ 2. $(B \vee C)$

La prima coppia di liste è:

$$\begin{array}{l|l} (A \wedge \neg B) & (B \vee C) \end{array}$$

Poiché '$(A \wedge \neg B)$' è una congiunzione appartenente alla lista di sinistra, si ottiene:

$$\begin{array}{l|l} A & (B \vee C) \\ \neg B & \end{array}$$

Siccome poi '(B∨C)', nella lista di destra, è una disgiunzione, si ha:

A	B
¬B	C

Infine, dato che '¬B', nella lista di sinistra, è una negazione, l'applicazione della regola 1 conduce a:

A	B
	C
	B

Nessun enunciato compare in entrambe le liste; inoltre, tutti gli enunciati di entrambe le liste sono atomici.
Quando tutti gli enunciati di ogni lista sono atomici e nessun enunciato atomico compare in entrambe le liste, diremo che il tentativo di attribuire il valore di verità TRUE agli enunciati della lista di sinistra e FALSE a quelli della lista di destra *ha avuto successo.* Ogni volta che ciò avviene, possiamo concludere che l'argomentazione originaria non è valida. Quando ogni lista è composta solo di enunciati atomici, e nessun enunciato compare in entrambe le liste, abbiamo trovato una situazione in cui le premesse sono vere, ma la conclusione è falsa. Considerando l'ultima coppia di liste ricavata dall'argomentazione precedente, notiamo che la situazione è:

V(A) = TRUE
V(B) = FALSE
V(C) = FALSE

perché A appartiene alla lista degli enunciati cui si vuole attribuire il valore di verità TRUE, mentre B e C appartengono alla lista degli enunciati cui si vuole attribuire il valore di verità FALSE.
Consideriamo un'altra argomentazione:

1. ¬A
2. (A∨B)
∴3. B

La prima coppia di liste ottenibile da questa argomentazione è:

¬A	B
(A∨B)	

Applicando la regola 1 otteniamo la seguente coppia di liste:

$(A \vee B)$	B
	A

Ma a questo punto come si può procedere? La regola 2 si applica solo a congiunzioni appartenenti alla lista di sinistra, mentre la regola 3 si applica solo a disgiunzioni appartenenti alla lista di destra. In questo caso abbiamo invece una disgiunzione nella lista di sinistra.
Se un enunciato compare nella lista di sinistra, ciò significa che gli si vuole attribuire il valore di verità TRUE. Affinché una disgiunzione valga TRUE occorre che almeno uno dei disgiunti valga TRUE. Rimane però da scegliere il disgiunto cui attribuire il valore di verità TRUE. Nel nostro caso, esistono due modi per far sì che '$(A \vee B)$' valga TRUE: si può attribuire il valore di verità TRUE al disgiunto 'A', oppure al disgiunto 'B'.
In altre parole, ci sono almeno due modi per attribuire il valore di verità TRUE ad '$(A \vee B)$'. Ciò significa che a questo punto dobbiamo creare due nuove coppie di liste, duplicando la coppia originaria e sostituendo la disgiunzione con un disgiunto diverso in ciascun duplicato:

1.	$(A \vee B)$	B		
		A		
2(a)	$(A \vee B)$	B	$(A \vee B)$	B
		A		A
2(b).	A	B	B	B
		A		A

Il tentativo di attribuire il valore di verità TRUE ad '$(A \vee B)$' si ramifica così in due nuovi tentativi: il primo rende vero '$(A \vee B)$' attribuendo il valore TRUE ad A, il secondo attribuendo il valore TRUE a B.
Si noti però che entrambi i tentativi contengono un enunciato che compare in tutte e due le liste: 'A' nel tentativo di sinistra, 'B' nel tentativo di destra. Perciò entrambi i tentativi falliscono, e l'argomentazione è quindi valida.
Aggiungiamo ora quattro regole nuove per la manipolazione delle liste:

4. Se nella lista di sinistra compare una disgiunzione, si creano due nuove coppie di liste, duplicando la coppia originaria. In una delle due nuove coppie si sostituisce la disgiunzione con il suo primo disgiunto, mentre nell'altra la si sostituisce con il suo secondo disgiunto.
5. Se nella lista di destra compare una congiunzione, si creano due nuove coppie di liste, duplicando la coppia originaria. In una delle due nuove coppie

si sostituisce la congiunzione con il suo primo congiunto, mentre nell'altra la si sostituisce con il suo secondo congiunto.
6. Se nella lista di sinistra compare un condizionale, si creano due nuove coppie di liste, duplicando la coppia originaria. In una delle due nuove coppie si sostituisce il condizionale con la negazione del suo antecedente, mentre nell'altra lo si sostituisce con il conseguente.
7. Se nella lista di destra compare un condizionale, lo si sostituisce con due enunciati: il suo conseguente e la negazione del suo antecedente.

La regola 6 si basa sul fatto che un condizionale vale TRUE se l'antecedente vale FALSE *o* se il conseguente vale TRUE. Viceversa, la regola 7 si basa sul fatto che un condizionale vale FALSE se il suo antecedente vale TRUE *e* se il suo conseguente vale FALSE. Per attribuire il valore di verità TRUE all'antecedente, si attribuisce il valore di verità FALSE alla negazione di tale antecedente.
Ogni volta che si incontra una regola che richiede una ramificazione in due tentativi, si continua sempre a lavorare sul primo tentativo, tenendo il secondo in sospeso, per riconsiderarlo più avanti. Secondo la terminologia informatica, ciò equivale a mettere *in pila* queste altre coppie di liste ottenute per ramificazione.
L'ALGORITMO DI WANG è ormai praticamente completo. Occorre solo aggiungere le due seguenti generalizzazioni per la valutazione di queste liste:

> Se fallisce ogni tentativo di attribuire il valore di verità TRUE agli enunciati delle liste di sinistra e FALSE a quelli delle liste di destra (inclusi i tentativi ottenuti per ramificazione), allora l'argomentazione è valida.
> Se anche un solo tentativo ha successo, l'argomentazione non è valida.

Abbiamo già definito il significato di "fallimento" e "successo" di un tentativo: un tentativo fallisce quando un enunciato compare in entrambe le liste, mentre ha successo quando in ogni lista compaiono solo enunciati atomici e nessun enunciato compare in entrambe le liste.
Siamo ora in grado di definire completamente l'algoritmo.

ALGORITMO DI WANG

1. INPUT un'argomentazione espressa in forma simbolica.
2. Creare due liste: mettere le premesse dell'argomentazione nella lista di sinistra e la conclusione nella lista di destra.
3. FOR ogni enunciato delle due liste

   (i) IF l'enunciato è una negazione
   THEN
   (a) Spostarlo nell'altra lista, eliminandone il segno di negazione più esterno.

(b) VERIFICARE le liste. (VERIFICARE è un sottoprogramma descritto nel seguito.)

(ii) IF l'enunciato è una congiunzione appartenente alla lista di sinistra THEN
(a) Sostituire tale congiunzione con i suoi due congiunti, elencati separatamente.
(b) VERIFICARE le liste.

(iii) IF l'enunciato è una disgiunzione appartenente alla lista di destra THEN
(a) Sostituire tale disgiunzione con i suoi due disgiunti, elencati separatamente.
(b) VERIFICARE le liste.

(iv) IF l'enunciato è una disgiunzione appartenente alla lista di sinistra THEN
(a) Creare due nuove coppie di liste duplicando la coppia originaria .
(b) Nella prima delle due nuove coppie, che costituirà il tentativo corrente, sostituire la disgiunzione con il suo primo disgiunto.
(c) VERIFICARE le liste.
(d) Nella seconda delle due nuove coppie, da mettere in pila per essere elaborata in seguito, sostituire la disgiunzione con il suo secondo disgiunto.
(e) VERIFICARE le liste.

(v) IF l'enunciato è una congiunzione appartenente alla lista di destra THEN
(a) Creare due nuove coppie di liste duplicando la coppia originaria.
(b) Nella prima delle due nuove coppie, che costituirà il tentativo corrente, sostituire la congiunzione con il suo primo congiunto.
(c) VERIFICARE le liste.
(d) Nella seconda delle due nuove coppie, da mettere in pila, sostituire la congiunzione con il suo secondo congiunto.
(e) VERIFICARE le liste.

(vi) IF l'enunciato è un condizionale appartenente alla lista di sinistra THEN
(a) Creare due nuove coppie di liste duplicando la coppia originaria.
(b) Nella prima delle due nuove coppie, che costituirà il tentativo corrente, sostituire il condizionale con la negazione del suo antecedente.
(c) VERIFICARE le liste.
(d) Nella seconda delle due nuove coppie, da mettere in pila, sostituire il condizionale con il suo conseguente.
(e) VERIFICARE le liste.

(vii) IF l'enunciato è un condizionale appartenente alla lista di destra THEN
(a) Sostituirlo con due enunciati: il suo conseguente e la negazione del suo antecedente.

(b) VERIFICARE le liste.

4. STOP

La gestione della pila e dei risultati in uscita è affidata al sottoprogramma VERIFICARE:

SOTTOPROGRAMMA VERIFICARE

1. IF nel tentativo corrente un enunciato compare in entrambe le liste
   THEN
   (a) Contrassegnare questo tentativo come "fallito".
   (b) IF è rimasta almeno una coppia di liste nella pila
       THEN
       (i) Assumere come tentativo corrente l'ultima coppia di liste aggiunta alla pila.
       (ii) Cancellare tale coppia dalla pila.
       (iii) Eseguire il prossimo passo del programma principale.
   (c) IF non è rimasta nessuna coppia di liste nella pila
       THEN
       (i) OUTPUT "Argomentazione valida".
       (ii) STOP.
2. IF nel tentativo corrente nessun enunciato compare in entrambe le liste
   THEN
   (a) IF gli enunciati di entrambe le liste sono tutti atomici
       THEN
       (i) OUTPUT "Argomentazione non valida".
       (ii) STOP.
   (b) IF non tutti gli enunciati delle due liste sono atomici
       THEN eseguire il prossimo passo del programma principale.

CONNETTIVI PRINCIPALI

Al passo 3, l'ALGORITMO DI WANG deve determinare se un enunciato è una negazione, una disgiunzione, una congiunzione o un condizionale. Come si può operare tale distinzione in modo automatico? Un enunciato è un condizionale se il suo connettivo principale è $\rightarrow$ (e analogamente per gli altri connettivi).
Il *connettivo principale* di un enunciato è il connettivo circondato dal minimo numero di parentesi: il concetto di connettivo principale si contrappone al concetto di sottoformula più interna (vedi Capitolo 5).
Per individuare il connettivo principale di un enunciato si può usare il seguente sottoprogramma:

SOTTOPROGRAMMA CONNETTIVO PRINCIPALE

1. INPUT un enunciato.
2. IF l'enunciato è costituito da un solo carattere
   THEN
   (a) L'enunciato è atomico e non ha un connettivo principale.
   (b) STOP.
3. IF il primo carattere è ¬
   THEN
   (a) Il connettivo principale è ¬.
   (b) STOP.
4. LET $n = 0$.
5. FOR ogni carattere dell'enunciato
   (a) IF il carattere è '('
       THEN LET $n = n + 1$
   (b) IF il carattere è ')'
       THEN LET $n = n - 1$
   (c) IF il carattere è '∨' e $n = 1$.
       THEN
       (i) Il connettivo principale è ∨.
       (ii) STOP.
   (d) IF il carattere è '∧' e $n = 1$
       THEN
       (i) Il connettivo principale è ∧.
       (ii) STOP.
   (e) IF il carattere è '→' e $n = 1$
       THEN
       (i) Il connettivo principale è →.
       (ii) STOP.

Si noti che i passi 5(a) e 5(b) di questo sottoprogramma realizzano il "primo contatore" dell'algoritmo CALCOLO DEL VALORE DI VERITÀ del Capitolo 5. Il "primo contatore" definisce il numero di parentesi da cui è racchiuso un carattere. Se l'enunciato non è atomico e non è una negazione, il minimo numero di parentesi da cui un carattere può essere racchiuso è 1; perciò la condizione '$n = 1$' al passo 5 indica che un carattere è circondato dal minimo numero di parentesi.

ALCUNI ESEMPI

Applichiamo l'ALGORITMO DI WANG a qualche argomentazione.
Si consideri l'argomentazione seguente:

1. A
2. (A→(B∧C))

∴ 3. C

Applicando l'ALGORITMO DI WANG, dopo i passi 1 e 2 si ottiene:

$$\begin{array}{l|l} A & C \\ (A \rightarrow (B \wedge C)) & \end{array}$$

Si noti che per il passo 3 non modifica gli enunciati atomici, come A e C; perciò il prossimo passo che viene eseguito è il 3(vi). Il risultato è una ramificazione in due tentativi. Qui e negli esempi successivi effettueremo in un solo passo la duplicazione della coppia di liste e le sostituzioni.

$$\begin{array}{r|l} A & C \\ \neg\, A & \end{array} \qquad\qquad \begin{array}{l|l} A & C \\ (B \wedge C) & \end{array}$$

Continuiamo a lavorare sulla prima coppia di liste, applicando ora il passo 3(i):

$$\begin{array}{r|l} A & C \\ & A \end{array}$$

FALLITO

Applicando VERIFICARE si scopre infatti che un enunciato compare in entrambe le liste. Prendiamo ora in considerazione l'altro tentativo ottenuto per ramificazione, che avevamo momentaneamente accantonato:

$$\begin{array}{l|l} A & C \\ (B \wedge C) & \end{array}$$

Applicando il passo 3(ii):

$$\begin{array}{l|l} A & C \\ B & \\ C & \end{array}$$

FALLITO
ARGOMENTAZIONE VALIDA

Verificando questa coppia di liste si scopre cioè che il tentativo fallisce. L'argomentazione risulta valida perché non ci sono altri tentativi da prendere in considerazione, cioè non ci sono altre coppie di liste nella pila.
Consideriamo un'altra argomentazione:

1. $\neg A$
2. $(A \rightarrow B)$

$\therefore$ 3. $\neg B$

Dapprima abbiamo:

$\neg A$	$\neg B$
$(A \rightarrow B)$	

poi:

	$\neg B$
$(A \rightarrow B)$	A

Ora si ha la seguente ramificazione:

$\neg A$	$\neg B$	B	$\neg B$
	A		A

Continuando ad operare sulla prima ramificazione si ottiene:

	$\neg B$
	A
	A

da cui:

B	A
	A

Poiché in ogni lista compaiono solo enunciati atomici e nessun enunciato compare in entrambe le liste, applicando VERIFICARE si conclude che l'argomentazione non è valida.

6.7 CONCLUSIONI

Riprendendo le tavole di verità, abbiamo descritto come costruirle per qualsiasi enunciato molecolare, comunque complesso, mediante l'algoritmo GENERATORE DI TAVOLE DI VERITÀ (che chiama CALCOLO DEL VALORE DI VERITÀ come sottoprogramma). Abbiamo poi definito un metodo che usa una tavola di verità per determinare se un'argomentazione è valida oppure no. I concetti fondamentali sono due: (1) si costruisce una tavola di verità riservando le colonne di sinistra agli enunciati atomici che compaiono nell'argomentazione e le colonne di destra alle premesse e alla conclusione; (2) le righe di questa tavola di verità

contengono l'informazione necessaria all'identificazione di eventuali situazioni in cui le premesse abbiano valore di verità TRUE e la conclusione abbia valore di verità FALSE. Questo metodo è applicato dall'algoritmo DETERMINAZIONE DI VALIDITÀ/NON VALIDITÀ, che chiama GENERATORE DI TAVOLE DI VERITÀ come proprio sottoprogramma. Benché abbastanza efficiente se eseguito da un calcolatore, questo algoritmo diventa rapidamente incontrollabile all'aumentare delle dimensioni del problema. L'ALGORITMO DI WANG è un procedimento più elegante ed efficiente per determinare la validità di un'argomentazione.

6.8 ESERCIZI

A. Per ogni argomentazione sotto elencata:
   (a) Dire quante righe contiene la relativa tavola di verità.
   (b) Scrivere i valori di verità nelle colonne degli enunciati atomici.

Esempio:

$(A \to (B \wedge \neg C))$
$(\neg B \wedge \neg C)$
$\therefore \neg A$

Risposta:
(a) Servono otto righe.
[Ci sono tre enunciati atomici distinti, A, B e C, e 2**3 = 8]

(b)

V(A)	V(B)	V(C)
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

1. $\neg(A \to B)$
   $\neg B$
   $\therefore A$
2. $(C \vee D)$
   $(\neg C \wedge (D \to B))$
   $\therefore (D \wedge B)$

3. $(A \wedge \neg A)$
B
$\therefore \neg B$
4. A
B
$\therefore (C \vee \neg C)$
5. $(A \rightarrow (B \wedge C))$
$(B \rightarrow D)$
A
$\therefore D$
6. $(A \rightarrow B)$
$(\neg B \vee C)$
$\neg A$
$\therefore \neg C$
7. $(\neg D \wedge (B \vee \neg C))$
$(\neg D \rightarrow B)$
$\therefore C$
8. $(B \rightarrow \neg (C \wedge D))$
$(C \wedge D)$
$\therefore \neg \neg \neg B$
9. $(A \rightarrow B)$
$(B \rightarrow C)$
$\therefore (A \rightarrow C)$
10. $(A \rightarrow B)$
$\therefore (\neg B \rightarrow \neg A)$
11. $(A \rightarrow B)$
$(A \rightarrow C)$
$\therefore (B \rightarrow C)$
12. $(A \vee B)$
$\neg A$
$\therefore B$

B. Esaminando i frammenti di parti destre di tavole di verità sotto riportati, rispondere alle seguenti domande: (1) L'argomentazione è valida o no? (2) Se non è valida, da quale riga lo si deduce?

1.

Prem. 1	Prem.2	Conclusione
0	1	1
1	1	1
0	0	0
1	1	0
1	0	0

2.

Prem. 1	Prem. 2	Prem. 3	Conclusione
1	0	1	1
0	0	0	0
1	1	1	1
0	1	1	1

3.

Prem. 1	Prem. 2	Conclusione
0	1	1
1	1	0
0	1	1
0	0	0
1	1	1
0	1	0
1	1	0
0	1	0

4.

Prem. 1	Prem. 2	Prem. 3	Prem. 4	Conclusione
0	1	1	1	0
1	1	1	0	0
1	1	1	1	0

C. Applicare GENERATORE DI TAVOLE DI VERITÀ alle argomentazioni dell'Esercizio A. Per ciascuna argomentazione, indicare se è valida oppure no.

D. I passi 2 e 3 di GENERATORE DI TAVOLE DI VERITÀ indicano che le dimensioni della tavola di verità necessaria per verificare un'argomentazione sono funzione (1) del numero di premesse; (2) del numero di enunciati atomici distinti che compaiono nelle premesse e nella conclusione. Calcolare le dimensioni, espresse in righe e colonne, delle tavole di verità necessarie per argomentazioni aventi:

1. 3 premesse e 2 enunciati atomici
2. 2 premesse e 3 enunciati atomici
3. 4 premesse e 3 enunciati atomici
4. 3 premesse e 5 enunciati atomici

E. L'algoritmo DETERMINAZIONE DI VALIDITÀ/NON VALIDITÀ prima compila per intero la tavola di verità (passo 2), poi esamina ogni riga, verificando se permette di concludere che l'argomentazione non è valida (passo 3). Si può però considerare un algoritmo alternativo, che compila solo la prima

riga e la esamina subito. Se la prima riga permette già di concludere che l'argomentazione non è valida, l'algoritmo fornisce in uscita il messaggio "Non valida" e termina. Altrimenti, l'algoritmo compila la riga successiva e ripete il processo.

Quali sono i vantaggi e gli svantaggi di questo metodo?

F. Applicare l'ALGORITMO DI WANG alle seguenti argomentazioni, per determinare quali sono valide e quali no.

1. $(A\wedge(C\vee D))$
   $\neg C$
   $(D\rightarrow E)$
   $\therefore E$
2. $(B\rightarrow\neg C)$
   $(A\vee(D\wedge B))$
   $\therefore (D\vee A)$
3. $(A\wedge C)$
   $(D\rightarrow A)$
   $(D\rightarrow E)$
   $\therefore (D\wedge E)$
4. $\neg(A\rightarrow\neg B)$
   $\neg(C\vee E)$
   $\therefore (C\rightarrow A)$
5. $(\neg A\wedge(C\rightarrow D))$
   $(D\rightarrow A)$
   $((F\wedge E)\rightarrow C)$
   $\therefore (\neg F\vee\neg E)$

G. Determinare una formula che, data un'argomentazione, calcoli la lunghezza massima di ciascuna lista dell'ALGORITMO DI WANG.
(*Suggerimento*: si consideri il numero degli enunciati atomici).

H. Determinare una formula che calcoli il massimo numero di ramificazioni che possono essere prodotte da un'applicazione dell'ALGORITMO DI WANG.

I. Scrivere un programma che applichi l'ALGORITMO DI WANG a una coppia di liste contenente, come connettivi principali, $\wedge$ e $\neg$ nella lista di sinistra e $\rightarrow$, $\neg$ e $\vee$ nella lista di destra. (Questa restrizione permette di non considerare le ramificazioni e la conseguente pila.)

J. Scrivere un programma che applichi l'ALGORITMO DI WANG, in versione completa, a un'argomentazione fornita in ingresso e stampi tutte le coppie di liste, in modo che ogni ramificazione si concluda o con il messaggio "Tentativo fallito", o con il messaggio "Tentativo riuscito, argomentazione non valida".

6.9 SUGGERIMENTI PER UNA REALIZZAZIONE SU CALCOLATORE

Il progetto di un programma che implementi l'ALGORITMO DI WANG richiede alcune considerazioni preliminari. In primo luogo occorre riservare una o più variabili per la memorizzazione della coppia di liste che costituisce il tentativo corrente. Occorre inoltre fare previsioni sulle coppie di liste da mettere in pila. In secondo luogo, bisogna progettare un sottoprogramma o una funzione che, ricevendo in ingresso un enunciato, determini se è un enunciato atomico, oppure una negazione, un condizionale, una congiunzione o una disgiunzione. Tale sottoprogramma può essere chiamato CONNETTIVO PRINCIPALE, poiché il suo scopo fondamentale è l'individuazione del connettivo principale di un enunciato. In terzo luogo, si deve progettare un sottoprogramma VERIFICARE, che verifichi le coppie di liste.
Le coppie di liste verranno memorizzate come vettori di stringhe nella maggior parte dei linguaggi di programmazione, oppure come liste vere e proprie in quei linguaggi, come il LISP e il LOGO, che sono dotati del tipo predefinito "lista". La lista destra e la lista sinistra del tentativo corrente possono essere chiamate, rispettivamente, LISTADEX e LISTASIN. Può essere utile avere due variabili, che chiameremo CONTDEX e CONTSIN, le quali memorizzino, istante per istante, il numero degli enunciati contenuti rispettivamente in LISTADEX e in LISTASIN.
Il sottoprogramma CONNETTIVO PRINCIPALE riceve in ingresso una stringa e fornisce in uscita

A se l'enunciato d'ingresso è atomico
¬ se è una negazione
∨ se è una disgiunzione
∧ se è una congiunzione
> se è un condizionale

Come già discusso nel capitolo precedente, è necessario sostituire i simboli non disponibili nel set dei caratteri ASNI (¬, ∧, ∨) con altri a scelta del programmatore.
Per manipolare le stringhe occorre anche definire delle funzioni, o dei sottoprogrammi, che effettuino le operazioni seguenti:

1. Rimuovere il segno di negazione, quando questo costituisce il primo carattere di una stringa (funzione RIMNEG).
2. Estrarre, da un enunciato ricevuto in ingresso, l'enunciato posto a sinistra del connettivo principale (funzione ENUNSIN).
3. Estrarre, da un enunciato ricevuto in ingresso, l'enunciato posto a destra del connettivo principale (funzione ENUNDEX).

Vediamo un esempio di applicazione di ciascuna di queste tre funzioni:

RIMNEG[¬ ¬(A∧B)] = ¬(A∧B)
ENUNSIN[((A∧B)∨(C∨D))] = (A∧B)
ENUNDEX[((A∧B)∨(C∨D))] = (C∨D)

Usando queste funzioni, l'inizio del programma, descritto in modo molto dettagliato nel nostro pseudolinguaggio di programmazione, potrebbe essere:

1. INPUT LISTASIN [premesse].
2. INPUT LISTADEX [conclusione].
3. LET CONTSIN = numero di enunciati di LISTASIN.
4. LET CONTDEX = 1 [perché all'inizio LISTADEX contiene solo la conclusione].
5. FOR ogni enunciato di LISTASIN FROM I = 1 TO CONTSIN
   (a) IF CONNETTIVO-PRINCIPALE(LISTASIN(I)) = ¬
       THEN
       (i) LET LISTADEX(CONDEX + 1) = RIMNEG(LISTASIN(I)).
       (ii) LET LISTASIN(I) = ' ' [spazio vuoto].
       (iii) LET CONTDEX = CONTDEX + 1.
       (iv) VERIFICARE la coppia di liste.
   (b) IF CONNETTIVO-PRINCIPALE(LISTASIN(I)) = ∧
       THEN
       (i) LET LISTASIN(CONTSIN + 1) = ENUNSIN(LISTASIN(I)).
       (ii) LET LISTASIN(I) = ENUNDEX(LISTASIN(I)).
       (iii) LET CONTSIN = CONTSIN + 1
       (iv) VERIFICARE la coppia di liste.

e analogamente per LISTADEX.
Le ramificazioni dovute a condizionali o a disgiunzioni di LISTASIN e a congiunzioni di LISTADEX richiedono alcune considerazioni aggiuntive. Viene messa in pila una coppia di liste per volta; occorrono dunque due pile, che chiameremo PILASIN e PILADEX: in PILASIN vengono memorizzate le liste sinistre messe in pila, mentre in PILADEX vengono memorizzate quelle destre. È opportuno che PILASIN e PILADEX siano matrici bidimensionali, in cui il primo indice precisa quando una lista è stata messa in pila (per prima, per seconda, ecc.), mentre il secondo indice individua un enunciato della lista. Ad esempio, PILASIN(2,3) individua il terzo enunciato della seconda lista sinistra messa in pila. Le liste vengono inserite nelle pile nello stesso ordine in cui vengono generate per ramificazione, ma, quando devono essere usate come tentativo corrente, vengono prelevate nell'ordine inverso: si preleva per prima l'ultima coppia di liste messa in pila. Questo metodo è detto "ultimo arrivato, primo servito", o, secondo la corrente terminologia inglese, "last in, first out" (LIFO). Esistono altre modalità di organizzazione, come, ad esempio, la coda, basata sul principio "primo arrivato, primo servito" ("first in, first out": FIFO); la pila però è più conveniente,

perché è probabile che le stringhe dell'ultima coppia di liste messa in pila siano più semplici di quelle della prima coppia; è quindi probabile che l'ultima coppia di liste permetta di dichiarare il fallimento di un tentativo dopo un ridotto numero di manipolazioni sulle due liste. È infine opportuno avere dei contatori che memorizzino il numero degli enunciati contenuti in ciascun livello di ognuna delle pile. Un vantaggio dell'ALGORITMO DI WANG consiste nel fatto che la sua realizzazione non richiede le enormi matrici a volte necessarie per DETERMINAZIONE DI VALIDITÀ/NON VALIDITÀ. Se non ci sono troppe premesse (diciamo non più di 6) né troppi enunciati atomici (diciamo non più di 10), è sufficiente che i vettori LISTASIN e LISTADEX abbiano 10 elementi e che PILASIN e PILADEX siano matrici 10 $\times$ 10.

7

Logica enunciativa: equivalenza logica, forme normali e notazione polacca

Dopo aver risolto uno dei problemi fondamentali della logica enunciativa, cioè la determinazione algoritmica della validità o non validità di un'argomentazione, affrontiamo ora altri argomenti di rilievo nella teoria della logica. Alcuni enunciati molecolari sono strutturati in modo da avere sempre valore di verità TRUE. Questi enunciati logicamente veri sono importanti di per sé, ma permettono anche di sviluppare altri metodi per determinare se un'argomentazione è valida o no. In questo capitolo esamineremo anche enunciati che, dal punto di vista della logica enunciativa, hanno lo stesso "significato". A causa dell'elevato numero di strutture che possono essere assunte anche da questi sinonimi verofunzionali (soprattutto se si usano così tanti connettivi diversi), ci si può chiedere se esistano dei modi standard di esprimere una proposizione con certi enunciati. Questo problema ci porta a trattare l'argomento delle forme normali, che, come vedremo nel Capitolo 14, sono molto utili per un'analisi al calcolatore di alcune proprietà degli enunciati, come la validità.
Esamineremo infine un sistema di notazione molto diverso da quello fin qui adottato: la notazione polacca, di utilizzazione assai diffusa nelle applicazioni informatiche. Tale sistema di notazione evita abilmente ogni ricorso alle parentesi e semplifica moltissimo alcune delle operazioni su stringhe precedentemente considerate (anche se ne complica alcune altre).

7.1 TAUTOLOGIE E ARGOMENTAZIONI

Abbiamo ammesso, e continueremo ad ammettere, che gli enunciati possano assumere solo uno dei due valori di verità TRUE e FALSE: se **P** è un enunciato, allora o V(**P**) = TRUE, o V(**P**) = FALSE. Per decidere se il valore di verità di un enunciato molecolare verofunzionale **P** è TRUE o FALSE, di solito si devono determinare i valori di verità delle sue parti atomiche. Esistono però due tipi di enunciati verofunzionali il cui valore di verità si può determinare anche senza co-

noscere i valori di verità delle parti atomiche; si tratta delle tautologie e delle contraddizioni.
Un enunciato **P** è una *tautologia* se e solo se V(**P**) = TRUE per ogni possibile combinazione dei valori di verità delle sue parti atomiche. Un enunciato è una *contraddizione* se e solo se è la congiunzione di un enunciato **Q** e della sua negazione ¬**Q**: è evidente che V(**Q**∧¬**Q**) = FALSE in ogni situazione. Una tautologia è dunque un enunciato sempre vero, mentre una contraddizione è un enunciato sempre falso.
Il terzo tipo di enunciato, ossia quello il cui valore di verità varia a seconda dei particolari valori di verità delle sue parti atomiche, è detto enunciato *contingente*. Poiché il valore di verità di una tautologia è TRUE e quello di una contraddizione è FALSE, indipendentemente dai valori di verità delle loro parti atomiche, si conclude che il valore di verità di una tautologia o di una contraddizione non varia al variare dei valori di verità delle parti atomiche.
Ad esempio, 'O piove o non piove' è una tautologia. Rappresentandola simbolicamente con

(A∨¬ A)

e considerando la relativa tavola di verità

V(A)	V(¬A)	V(A∨¬A)
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE

si veda che V(A∨¬A) = TRUE, indipendentemente.da V(A) e V(¬A). 'O piove o non piove' ha cioè valore di verità TRUE indipendentemente dal fatto che piova o meno: l'enunciato è vero qualunque tempo faccia, quindi non fornisce nessuna informazione sullo stato del tempo.
Si consideri ora l'enunciato 'Piove e non piove'. Si tratta di una contraddizione; infatti un esame della relativa tavola di verità mostra che ha sempre valore di verità FALSE:

V(A)	V(¬A)	V(A∧¬A)
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE

Neppure questo enunciato può fornire informazioni sullo stato del tempo, perché è sempre falso, sia se piove, sia se non piove.
In generale, si afferma che le tautologie e le contraddizioni non forniscono informazioni: solo gli enunciati contingenti ne forniscono. L'enunciato 'Piove, ma io ho l'ombrello' è contingente.

Se lo rappresentiamo simbolicamente con

$(A \wedge H)$

la sua tavola di verità è:

V(A)	V(H)	V(A∧H)
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

Questo enunciato, dunque, a volte è vero (quando piove *e* io ho l'ombrello) e a volte è falso (quando *non* piove *o* quando io *non* ho l'ombrello).
Poiché il fatto che un enunciato sia una tautologia o una contraddizione dipende dalla sua struttura e dalle proprietà delle funzioni logiche di verità, è evidente che nessun enunciato atomico può essere una tautologia o una contraddizione: nella logica enunciativa, tutti gli enunciati atomici sono contingenti.

ARGOMENTAZIONI E CONDIZIONALI CORRISPONDENTI

Le tautologie possono essere usate per determinare se un'argomentazione è valida o no. Ogni argomentazione è costituita da un insieme di premesse e da una conclusione. Consideriamo dapprima un'argomentazione con una sola premessa **P** e una conclusione **Q**:

1. **P**

∴ 2. **Q**

Questa argomentazione è valida se e solo se è impossibile che **Q** abbia valore di verità FALSE quando **P** vale TRUE; ma queste sono esattamente le circostanze in cui l'enunciato

$(\mathbf{P} \rightarrow \mathbf{Q})$

è una tautologia.
Consideriamo ora un'argomentazione con due premesse **P** e **Q** e una conclusione **R**:

1. **P**
2. **Q**

∴ 3. **R**

Questa argomentazione è valida se e solo se è impossibile che **R** abbia valore di verità FALSE quando **P** e **Q** valgono entrambi TRUE; in altre parole, questa argomentazione è valida se e solo se l'enunciato

$$((\mathbf{P} \wedge \mathbf{Q}) \rightarrow \mathbf{R})$$

è una tautologia.
In generale, ad ogni argomentazione con n premesse $\mathbf{P}_1$, $\mathbf{P}_2$, ..., $\mathbf{P}_n$ e una conclusione **Q**:

$$\begin{array}{rl} & 1.\ \mathbf{P}_1 \\ & 2.\ \mathbf{P}_2 \\ & \vdots \\ & n.\ \mathbf{P}_n \\ \therefore & n+1.\ \mathbf{Q} \end{array}$$

corrisponde un enunciato condizionale **R** il cui antecedente è la congiunzione delle premesse e il cui conseguente è la conclusione:

$$(\mathbf{R}) \qquad (((\ldots((\mathbf{P}_1 \wedge \mathbf{P}_2) \wedge \mathbf{P}_3) \wedge \ldots) \wedge \mathbf{P}_n) \rightarrow \mathbf{Q})$$

Fra un'argomentazione e il condizionale ad essa corrispondente intercorre la seguente relazione:

> Un'argomentazione è valida se e solo se il condizionale ad essa corrispondente è una tautologia.

7.2 EQUIVALENZA LOGICA

Può capitare che due enunciati diversi abbiano valore di verità TRUE o FALSE esattamente nelle stesse situazioni, cioè che, per ogni combinazione dei valori di verità delle loro parti atomiche, entrambi gli enunciati abbiano lo stesso valore di verità. Un semplice esempio può essere costituito da un enunciato atomico **P** e dalla sua doppia negazione $\neg\neg\mathbf{P}$: entrambi hanno la stessa parte atomica **P**, e risulta V(**P**) = TRUE se e solo se V($\neg\neg\mathbf{P}$) = TRUE. Diciamo allora che **P** e $\neg\neg\mathbf{P}$ sono *logicamente equivalenti.*
Come ulteriore esempio consideriamo un condizionale $(\mathbf{P} \rightarrow \mathbf{Q})$ e il suo *contrapposto* $(\neg\mathbf{Q} \rightarrow \neg\mathbf{P})$. Esaminiamo le corrispondenti tavole di verità (in cui rappresentiamo TRUE e FALSE rispettivamente con 1 e 0):

V(**P**)	V(**Q**)	V(**P**→**Q**)
0	0	1
0	1	1
1	0	0
1	1	1

V(**P**)	V(**Q**)	V(¬**P**)	V(¬**Q**)	V(¬**Q**→¬**P**)
0	0	1	1	1
0	1	1	0	1
1	0	0	1	0
1	1	0	0	1

Notiamo che V(**P**→**Q**) = V(¬**Q**→¬**P**) per ogni possibile combinazione dei valori di verità di **P** e di **Q**; perciò (**P**→**Q**) e (¬**Q**→¬**P**) sono logicamente equivalenti. In generale, due enunciati **P** e **Q** sono *logicamente equivalenti* se e solo se V(**P**) = V(**Q**) per ogni possibile combinazione dei valori di verità delle loro parti atomiche.
Due enunciati come '(A∨¬A)' e '(B∨¬B)' sono logicamente equivalenti? Non hanno enunciati atomici in comune, quindi si potrebbe pensare che non lo siano. Tuttavia entrambi sono tautologie, quindi entrambi hanno sempre valore di verità TRUE; dunque V(A∨¬A) = V(B∨¬B) per ogni possibile combinazione dei valori di verità delle loro parti atomiche; perciò i due enunciati sono logicamente equivalenti. Analogamente, sono logicamente equivalenti i due enunciati '(A∧¬A)' e '(B∧¬B)', perché sono entrambi contraddizioni e quindi valgono sempre FALSE. In generale, tutte le tautologie sono fra loro equivalenti, mentre è ovvio che nessuna tautologia è equivalente a qualche contraddizione.
Fra le tautologie e l'equivalenza logica intercorre un'altra relazione importante. Si consideri un bicondizionale (**P**↔**Q**); in base alla sua tavola di verità, V(**P**↔**Q**) = TRUE se e solo se V(**P**) = V(**Q**), cioè un bicondizionale vale TRUE se e solo se la sua parte sinistra e la sua parte destra hanno lo stesso valore di verità. Questo fatto suggerisce il seguente principio:

> Un enunciato **P** è logicamente equivalente a un enunciato **Q** se e solo se (**P**↔**Q**) è una tautologia.

Come esempio, riconsideriamo (**P**→**Q**) e (¬**Q**→¬**P**). Anziché costruire due tavole di verità separate per dimostrare che sono logicamente equivalenti, potremmo combinare le due tavole, costruendo solo la tavola di verità del bicondizionale

((**P**→**Q**) ↔ (¬**Q**→¬**P**))

per dimostrare poi che quest'ultimo enunciato è una tautologia.
Esistono diverse equivalenze logiche importanti; molte di esse saranno lasciate co-

me esercizio, ma tre meritano di essere trattate separatamente in questa sede. Le prime due sono le cosiddette *leggi di De Morgan*:

1. $(\neg(\mathbf{P}\wedge\mathbf{Q}) \leftrightarrow (\neg\mathbf{P}\vee\neg\mathbf{Q}))$
2. $(\neg(\mathbf{P}\vee\mathbf{Q}) \leftrightarrow (\neg\mathbf{P}\wedge\neg\mathbf{Q}))$

Gli enunciati (1) e (2) sono tautologie, ossia in essi la parte sinistra è logicamente equivalente alla parte destra. È importante capire bene che cosa ciò significhi. La legge di De Morgan (1) afferma che la negazione di una congiunzione è logicamente equivalente a una disgiunzione, e più precisamente alla disgiunzione delle negazioni dei congiunti originari. Analogamente, la legge di De Morgan (2) afferma che la negazione di una disgiunzione è logicamente equivalente a una congiunzione, e più precisamente alla congiunzione delle negazioni dei disgiunti originari. La parte sinistra della legge (1) è anche logicamente equivalente a (**P** NAND **Q**); perciò (**P** NAND **Q**) è logicamente equivalente a una disgiunzione. Si verifichino tutti questi fatti costruendo le tavole di verità appropriate.
Un'altra equivalenza logica importante è detta *esportazione*. L'enunciato

$$(((\mathbf{P}\wedge\mathbf{Q}) \rightarrow \mathbf{R}) \leftrightarrow (\mathbf{P}\rightarrow (\mathbf{Q}\rightarrow\mathbf{R})))$$

è una tautologia; perciò le due parti del bicondizionale sono logicamente equivalenti.
Queste regole possono essere particolarmente utili nella stesura di programmi per calcolatore, per esprimere in modo efficiente delle condizioni che devono essere verificate. Ad esempio, alcuni calcolatori non ammettono istruzioni condizionali del tipo

```
IF  P∧Q
    THEN esegui X
```

Se **P** vale FALSE e se a **Q** non è ancora stato assegnato un valore, si può avere un errore durante l'esecuzione. Questo inconveniente si può evitare usando l'istruzione logicamente equivalente

```
IF  P
    THEN IF Q
      THEN esegui X
```

Se **P** vale FALSE, l'istruzione non viene eseguita e quindi un eventuale valore non lecito di **Q** non viene preso in considerazione.

7.3 FORME NORMALI

Si supponga di voler dimostrare qualche affermazione riguardante tutti gli enunciati: sarebbe utile disporre di qualche modo uniforme di esprimere gli enunciati, cioè di una cosiddetta *forma normale*. Le forme normali sono concepite in modo che ogni enunciato sia logicamente equivalente a un enunciato espresso in forma normale. In questo paragrafo descriveremo la *forma normale congiuntiva*; un'altra forma normale importante, quella *disgiuntiva*, è trattata nell'Esercizio G. Per poter definire la forma normale congiuntiva è utile premettere la definizione di *disgiunzione elementare*:

1. Se **P** è un enunciato atomico o la negazione di un enunciato atomico
   allora **P** è una disgiunzione elementare.
2. Se **P** e **Q** sono disgiunzioni elementari
   allora (**P**∨**Q**) è una disgiunzione elementare.
3. Nient'altro è una disgiunzione elementare.

Ad esempio,

A
$\neg A$
$\neg B$
$(A \vee B)$
$(A \vee \neg A)$
$((A \vee B) \vee C)$
$((A \vee B) \vee (D \vee \neg E))$

sono tutte disgiunzioni elementari.
Possiamo ora definire quando un enunciato è espresso in forma normale congiuntiva (FNC):

1. Se **P** è una disgiunzione elementare
   allora **P** è in FNC.
2. Se **P** e **Q** sono entrambi in FNC
   allora (**P**∧**Q**) è in FNC.
3. Nient'altro è in FNC.

Sono, ad esempio, in FNC tutte le disgiunzioni elementari sopra elencate, nonché gli enunciati seguenti:

$((A \vee B) \wedge (\neg A \vee B))$
$(A \wedge \neg A)$
$(((A \vee B) \wedge (A \vee C)) \wedge \neg B)$

Si noti che un enunciato in FNC è sempre una *congiunzione multipla di disgiunzioni*, purché si convenga di considerare gli enunciati atomici isolati e le loro negazioni come casi degeneri di congiunzioni e disgiunzioni.
Esistono diversi algoritmi per convertire qualsiasi enunciato **P** in un enunciato equivalente **Q** espresso in FNC; uno di questi usa le tavole di verità:

ALGORITMO FNC-1

1. INPUT un enunciato **P**.
2. Costruire la tavola di verità di **P** applicando GENERATORE DI TAVOLE DI VERITÀ.
3. IF **P** è una tautologia
   THEN LET **Q** = (**R**∨¬**R**), ove **R** è il primo enunciato atomico di **P** in ordine alfabetico.
4. IF **P** non è una tautologia
   THEN
   (a) FOR ogni riga 'RIGA' della tavola di verità tale che V(**P**) = 0
      (i) Costruire un insieme ATOM-NEG(RIGA) di enunciati atomici e di loro negazioni, operando nel modo seguente:
         (1) FOR ogni parte atomica **R**
            (a) IF V(**R**) = 1
                  THEN inserire ¬**R** nell'insieme
            (b) IF V(**R**) = 0
                  THEN inserire **R** nell'insieme
      (ii) LET DISG-ELEM(RIGA) = disgiunzione degli elementi di ATOM-NEG(RIGA).
         (Si noti che DISG-ELEM(RIGA) è una disgiunzione elementare.)
   (b) LET **Q** = congiunzione di tutte le disgiunzioni elementari DISG-ELEM(RIGA).
5. OUTPUT l'enunciato in FNC **Q**.
6. STOP.

Un altro metodo per convertire un enunciato in FNC non richiede l'uso di tavole di verità:

ALGORITMO FNC-2

1. INPUT un enunciato **P**.
2. Sostituire tutte le sottoformule del tipo (**R**↔**S**) con enunciati del tipo ((**R**→**S**) ∧ (**S**→**R**)). (*Domanda*: quale principio permette di effettuare questa sostituzione?)
3. Sostituire tutte le sottoformule di tipo (**R**→**S**) con enunciati di tipo (¬**R**∨**S**).

4. Ripetere i passi seguenti fino al momento in cui gli enunciati negati siano tutti atomici:
   (a) Sostituire tutte le sottoformule del tipo ¬(**R**∧**S**) con enunciati del tipo (¬**R**∨¬**S**).
   (b) Sostituire tutte le sottoformule del tipo ¬(**R**∨**S**) con enunciati del tipo (¬**R**∧¬**S**).
   (c) Sostituire tutte le sottoformule del tipo ¬¬**R** con enunciati del tipo **R**.
5. Ripetere i passi seguenti finché non si raggiunge un enunciato in FNC:
   (a) Sostituire tutte le sottoformule del tipo (**R**∨(**S**∧**T**)) con enunciati del tipo ((**R**∨**S**) ∧ (**R**∨**T**)).
   (b) Sostituire tutte le sottoformule del tipo ((**R**∧**S**)∨**T**) con enunciati del tipo ((**R**∨**T**) ∧ (**S**∨**T**)).
6. LET **Q** = risultato.
7. OUTPUT l'enunciato in FNC **Q**.
8. STOP.

La FNC permette di identificare facilmente le tautologie; infatti, se si usa l'algoritmo FNC-2 per trasformare un enunciato in un altro, logicamente equivalente, espresso in FNC, si può determinare se si tratta di una tautologia senza dover usare tavole di verità, in base al principio seguente:

> Un enunciato in FNC è una tautologia se e solo se ogni congiunto contiene sia un enunciato atomico, sia la sua negazione.

Si consideri ad esempio l'enunciato

((A∧(A→B))→B)

Determineremo l'enunciato logicamente equivalente in FNC, usando entrambi gli algoritmi, e verificheremo se si tratta di una tautologia.

Algoritmo FNC-1: costruiamo dapprima la tavola di verità, nel modo ormai noto. Da essa si ricava subito che l'enunciato è una tautologia. Perciò un enunciato in FNC ad esso equivalente è

(A∨¬A)

Algoritmo FNC-2:

Passo 2: non è applicabile al nostro esempio.
Passo 3: l'enunciato è trasformato prima in
  ((A∧(¬A∨B)) →B)
 poi in
  (¬(A∧(¬A∨B))∨B)
Passo 4(a): ((¬A∨¬(¬A∨B))∨B)

Passo 4(b): $((\neg A \vee (\neg\neg A \wedge \neg B)) \vee B)$
Passo 4(c): $((\neg A \vee (A \wedge \neg B)) \vee B)$
Passo 5(a): $(((\neg A \vee A) \wedge (\neg A \vee \neg B)) \vee B)$
Passo 5(b): $(((\neg A \vee A) \vee B)) \wedge ((\neg A \vee \neg B) \vee B))$

Quest'ultimo enunciato è in FNC, perché è una congiunzione di disgiunzioni elementari; infatti ogni congiunto è una disgiunzione, eventualmente multipla, di enunciati atomici e di negazioni di enunciati atomici. Inoltre, poiché ogni congiunto contiene sia un enunciato atomico, sia la sua negazione, si conclude che l'enunciato originario è una tautologia.
Il fatto che i due algoritmi abbiano prodotto due enunciati in FNC diversi fra loro potrebbe lasciare perplessi. In generale, esistono infiniti enunciati in FNC equivalenti a un enunciato dato (lo si dimostri); l'unica cosa importante è che l'enunciato sia in FNC e sia logicamente equivalente all'enunciato originario.

7.4 COERENZA E SODDISFACIBILITÀ

Diremo *soddisfacibile* un enunciato che assuma il valore di verità TRUE in alcune situazioni, cioè in corrispondenza di alcuni valori di verità delle sue componenti atomiche.
Una contraddizione è un enunciato il cui valore di verità è sempre FALSE, indipendentemente dai valori di verità delle sue componenti atomiche; per questo motivo le contraddizioni non sono soddisfacibili. Le tautologie e gli enunciati contingenti, invece, potendo assumere il valore di verità TRUE, sono soddisfacibili. Il concetto di soddisfacibilità permette di analizzare alcune importanti proprietà di insiemi di enunciati, e costituisce inoltre un altro potente strumento per determinare la validità delle argomentazioni.
Prima di trattare in generale il concetto di soddisfacibilità, esaminiamo alcuni semplici esempi:

(A) 1. La macchina parte solo se nel serbatoio c'è benzina.
    2. Nel serbatoio c'è benzina, ma la macchina non parte.

La rappresentazione simbolica seguente, con ovvio significato delle abbreviazioni, chiarisce meglio la struttura di questi due enunciati:

(A) 1. $(M \rightarrow B)$
    2. $(B \wedge \neg M)$

È facile dimostrare che sono soddisfacibili sia (A1) (basta assumere V(M) = FALSE), sia (A2). Ci chiediamo ora: sono simultaneamente soddisfacibili? In altre parole: esiste uno *stesso* assegnamento di valori di verità a tutte le componenti atomiche che renda veri *entrambi* gli enunciati molecolari? In questo esempio, l'asse-

gnamento V(M) = FALSE e V(B) = TRUE fa assumere il valore di verità TRUE sia ad (A1) che ad (A2).
Vediamo ora un esempio in cui gli enunciati sono soddisfacibili singolarmente, ma non simultaneamente.

(B) 1. Se i granchi pizzicano, non è il momento di nuotare.
2. È il momento di nuotare, ma i granchi pizzicano.

La rappresentazione simbolica è:

(B) 1. $(G \rightarrow \neg N)$
2. $(N \wedge G)$.

Si noti che in questo esempio

$V(G \rightarrow \neg N) = $ TRUE quando $V(G) = $ FALSE e
$V(N \wedge G) = $ TRUE quando $V(N) = V(G) = $ TRUE

perciò ciascun enunciato è soddisfacibile. Non esiste però uno stesso assegnamento di valori a V(G) e V(N) per il quale $V(G \rightarrow \neg N)$ e $V(N \wedge G)$ abbiano entrambi valore di verità TRUE contemporaneamente.
Queste considerazioni si possono estendere facilmente al caso di un numero qualsiasi di enunciati.

Se esiste uno stesso assegnamento di valori di verità a tutte le componenti atomiche di tutti gli enunciati di un insieme, tale che tutti gli enunciati dell'insieme assumano valore di verità TRUE, allora gli enunciati dell'insieme si dicono *simultaneamente soddisfacibili.*

Questo concetto si può usare per verificare la validità di un'argomentazione. Se un'argomentazione è valida, la conclusione è vera quando le premesse sono tutte vere; perciò, aggiungendo alle premesse la negazione della conclusione, se l'argomentazione è valida si ottiene un insieme di enunciati non simultaneamente soddisfacibili. Ora, dimostrare che un insieme di enunciati non è simultaneamente soddisfacibile è, sotto alcuni aspetti, più facile che dimostrare che un'argomentazione è valida, soprattutto nel caso di insiemi finiti di enunciati.
Un metodo semplice per dimostrare che gli enunciati di un insieme non sono simultaneamente soddisfacibili consiste nel fare di tutti un'unica congiunzione; si trasforma poi tale congiunzione in forma normale *disgiuntiva* (vedi Esercizio G): se ciascun disgiunto contiene una contraddizione, gli enunciati dell'insieme non sono simultaneamente soddisfacibili.
Finora abbiamo usato il concetto di simultanea soddisfacibilità per verificare la validità di un'argomentazione. È possibile invertire questo criterio e usare il concetto di validità di un'argomentazione per verificare se gli enunciati di un insieme sono simultaneamente soddisfacibili? Ricordiamo ancora una volta che un'argo-

mentazione è valida se la conclusione è vera quando le premesse sono vere. Se dunque la conclusione di un'argomentazione *valida* è una contraddizione, e quindi non può essere vera, rovesciando il ragionamento si può affermare che non tutte le premesse possono essere vere, o, più precisamente, che l'insieme delle premesse non è simultaneamente soddisfacibile.
Riassumiamo ora le considerazioni precedenti, introducendo un po' di terminologia nuova.

Se gli enunciati di un insieme sono usati come premesse di un'argomentazione valida, la cui conclusione è una contraddizione, allora quell'insieme di enunciati si dice *contraddittorio*.
Se non esiste un'argomentazione valida avente come premesse gli enunciati dell'insieme e come conclusione una contraddizione, allora l'insieme di enunciati si dice *coerente*.

In altre parole, se nessuna contraddizione è una conclusione valida per un insieme di premesse, tale insieme è coerente.
I concetti su cui si basa il presente paragrafo sono: contraddizione; coerenza, contraddittorietà e simultanea soddisfacibilità di un insieme di enunciati; validità di un'argomentazione. Vediamo ora alcune correlazioni fra questi concetti.

1. Un'argomentazione valida può avere per conclusione una contraddizione
   se e solo se
   le premesse sono contraddittorie
   se e solo se
   le premesse non sono simultaneamente soddisfacibili.
2. Le premesse di un'argomentazione e la negazione della conclusione non sono simultaneamente soddisfacibili
   se e solo se
   l'insieme che esse formano è contraddittorio
   se e solo se
   l'argomentazione è valida.
3. L'insieme delle premesse di un'argomentazione è contraddittorio
   se e solo se
   l'insieme delle premesse non è simultaneamente soddisfacibile
   se e solo se
   qualunque argomentazione con tali premesse è valida.

Giustifichiamo quest'ultima affermazione. Se le premesse di un'argomentazione non sono simultaneamente soddisfacibili, non esiste nessuna situazione in cui le premesse siano tutte vere e la conclusione sia falsa; perciò l'argomentazione è valida.
Per le applicazioni pratiche, la relazione più importante è la (2). Ad esempio, per dimostrare che l'argomentazione

1. $(A \vee B)$
2. $\neg A$

$\therefore$ 3. B

è valida, mettiamo in congiunzione tutte le premesse e la negazione della conclusione:

$$(((A \vee B) \wedge \neg A) \wedge \neg B)$$

Trasformando questa congiunzione in forma normale disgiuntiva, secondo le tecniche dell'Esercizio G, otteniamo:

$$(((A \wedge \neg A) \vee (B \wedge \neg A)) \wedge \neg B)$$

e quindi:

$$(((A \wedge \neg A) \wedge \neg B) \vee ((B \wedge \neg A) \wedge \neg B))$$

Ogni disgiunto contiene una contraddizione; perciò l'argomentazione originaria è valida.

7.5 NOTAZIONE POLACCA

L'uso delle parentesi, necessario per evitare ambiguità, è però gravoso. Spesso, introducendo convenzioni di precedenza fra i connettivi logici, si può ridurre il numero delle parentesi, rendendo così più leggibili le formule. Un sistema di notazione piuttosto diverso permette però di evitare completamente l'uso delle parentesi.

La notazione fin qui usata è detta *infissa*, perché i connettivi verofunzionali binari ($\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, NAND, NOR, XOR) sono posti fra le lettere proposizionali. Il segno di negazione, al contrario, viene prefisso all'enunciato cui si riferisce. Come vedremo, se si usa una notazione *prefissa* per tutti i connettivi, le parentesi non servono.

Il tipo di notazione prefissa più usato in logica è detto *notazione polacca*. Nella notazione polacca i connettivi non vengono rappresentati con i simboli da noi usati, bensì con lettere maiuscole corsive:

$\neg$ *N* (''Negazione'')
$\vee$ *A* (''Alternativa'')
$\wedge$ *K* (congiunzione, dal tedesco ''Konjunktion'')
$\rightarrow$ *C* (''Condizionale'' materiale)
$\leftrightarrow$ *E* (''Equivalenza'')

Nella notazione polacca, il concetto di enunciato ben formato si definisce nel modo seguente:

1. Nella notazione polacca, ogni enunciato atomico è ben formato.
2. Se **P** e **Q** sono ben formati secondo la notazione polacca, allora lo sono anche

   *N***P**
   *A***PQ**
   *K***PQ**
   *C***PQ**
   *E***PQ**

3. Nient'altro è ben formato nella notazione polacca.

I cinque enunciati della clausola 2 della definizione corrispondono a

$\neg \mathbf{P}$
$(\mathbf{P} \vee \mathbf{Q})$
$(\mathbf{P} \wedge \mathbf{Q})$
$(\mathbf{P} \rightarrow \mathbf{Q})$
$(\mathbf{P} \leftrightarrow \mathbf{Q})$

nella notazione infissa.
Consideriamo alcuni altri enunciati espressi in entrambe le notazioni:

	Infissa	Polacca
(1)	$(\mathbf{P} \wedge (\mathbf{Q} \wedge \mathbf{R}))$	*K***P***K***QR**
(2)	$((\mathbf{P} \wedge \mathbf{Q}) \wedge \mathbf{R})$	*KK***PQR**

Nell'esempio (1), la prima e la seconda *K* corrispondono rispettivamente alla prima e alla seconda ∧. La stringa in notazione polacca si può leggere nel modo seguente: congiunzione di **P** con (congiunzione di **Q** con **R**).
Nell'esempio (2), la prima *K* corrisponde alla seconda ∧, mentre la seconda *K* corrisponde alla prima ∧. La stringa in notazione polacca si può leggere nel modo seguente: congiunzione di (congiunzione di **P** con **Q**) con **R**.
Come si può vedere, nella notazione polacca le parentesi non sono necessarie (anche se in alcuni casi possono essere utili), perché le stringhe in notazione polacca non sono ambigue. Questo fatto si può forse chiarire con i seguenti esempi:

	Infissa	Polacca
(3)	$(\mathbf{P} \vee (\mathbf{Q} \wedge \mathbf{R}))$	*A***P***K***QR**
(4)	$((\mathbf{P} \vee \mathbf{Q}) \wedge \mathbf{R})$	*KA***PQR**

Come esercizio, si verifichi l'equivalenza fra le stringhe seguenti:

$(\mathbf{P} \lor (\mathbf{Q} \to \mathbf{R}))$	$A\mathbf{P}C\mathbf{QR}$
$(\neg(\mathbf{P} \land \mathbf{Q}) \leftrightarrow (\neg \mathbf{P} \lor \neg \mathbf{Q}))$	$ENK\mathbf{PQ}AN\mathbf{P}N\mathbf{Q}$

Esiste un algoritmo semplice ed elegante per determinare se una stringa in notazione polacca è ben formata; esso mostra, fra l'altro, che la soluzione di alcuni problemi non richiede che si prenda sempre in considerazione il significato dei simboli.

ALGORITMO PBF

1. INPUT una stringa **P** espressa in notazione polacca.
2. FOR ogni lettera proposizionale **Q** di **P**
   (a) LET RANGO(**Q**) = 1.
3. LET RANGO(*N*) = 0.
4. LET RANGO(*A*) = —1.
5. LET RANGO(*K*) = —1.
6. LET RANGO(*C*) = —1.
7. LET RANGO(*E*) = —1.
8. LET SOMMA = 0.
9. FOR ogni simbolo **S** di **P**, da destra a sinistra
   (a) LET SOMMA = SOMMA + RANGO(**S**).
   (b) IF SOMMA $\leqslant$ 0
      THEN
      (i) OUTPUT "Non ben formato."
      (ii) STOP.
10. IF SOMMA = 1
    THEN OUTPUT "Ben formato."
11. IF SOMMA > 1
    THEN OUTPUT "Non ben formato."
12. STOP.

Sarebbero utili altri due algoritmi: uno per convertire da notazione infissa a notazione polacca e uno per effettuare la conversione inversa. Consideriamo il primo. In prima approssimazione, vogliamo sostituire ogni $\neg \mathbf{P}$ con $N\mathbf{P}$, ogni $(\mathbf{P} \lor \mathbf{Q})$ con $A\mathbf{PQ}$, eccetera. Consideriamo ad esempio $((\mathbf{P} \land \mathbf{Q}) \lor (\mathbf{R} \land \mathbf{S}))$. Da dove cominciamo? Potremmo lavorare dall'interno verso l'esterno, cominciando cioè dalle sottoformule più interne; otterremmo così prima l'ibrido $(K\mathbf{PQ} \lor K\mathbf{RS})$, poi $AK\mathbf{PQ}K\mathbf{RS}$. In alternativa, si potrebbe procedere nel modo seguente: prima si identifica il connettivo principale (in questo caso $\lor$), per formare l'ibrido $A(\mathbf{P} \land \mathbf{Q})\ (\mathbf{R} \land \mathbf{S})$; poi si applica il procedimento ad ogni enunciato molecolare rimasto, identificandone il

connettivo principale e traducendolo in notazione polacca; si ottiene così *AK***PQ***K***RS**.
Il seguente algoritmo applica quest'ultimo procedimento.

ALGORITMO INFISSA-POLACCA

1. INPUT un enunciato **P** in notazione infissa.
2. IF **P** è atomico
   THEN
      (a) OUTPUT **P**.
      (b) STOP.
3. IF **P** non è atomico
      THEN
      (a) Applicare l'algoritmo CONNETTIVO PRINCIPALE (Capitolo 6).
      (b) LET ☆ = connettivo principale di **P**.
4. IF **P** ha la forma ¬**Q** (cioè IF ☆ = ¬).
      THEN trasformare **P** in *N***Q**.
5. IF **P** ha la forma (**Q**☆**R**)
      THEN trasformare **P** in *A***QR**, *K***QE**, *C***QR** o *E***QR**, conformemente al valore di ☆.
6. Ripetere i passi da 3 a 5 per ogni sottoformula dell'enunciato in corso di traduzione, fino ad ottenere una stringa priva di connettivi infissi.
7. OUTPUT il risultato.
8. STOP.

La notazione polacca, oltre a permettere l'eliminazione delle parentesi, rende estremamente facile l'identificazione del connettivo principale, e quindi semplifica molto l'applicazione dell'ALGORITMO DI WANG. La facilità di individuazione del connettivo principale agevola anche l'identificazione della struttura di un enunciato espresso in notazione polacca. Si consideri, ad esempio,

(5) *CKN*B*C*G B*N*G

Il connettivo principale è sempre il *primo*; perciò il connettivo principale dell'enunciato (5) è *C*. Poiché *C* è un connettivo binario, l'enunciato (5) ha la forma *C***PQ**. Che cosa sono **P** e **Q**?
Poiché **P** deve seguire immediatamente *C*, **P** comincia con *K*. Siccome *K* è a sua volta un connettivo binario, **P** deve avere la forma *K***RS**, ove **R** comincia con *N*. Ma *N* è un connettivo unario; perciò **R** è *N*B. Allora **S** dev'essere *C*GB, e quindi **Q** è *N*G. Evidenziamo questa struttura usando le parentesi:

C(*K*(*N*B)(*C*GB))(*N*G)

equivalente a

$$((\neg B \wedge (G \rightarrow B)) \rightarrow \neg G)$$

in notazione infissa.

7.6 CONCLUSIONI

Abbiamo cominciato questo capitolo considerando gli enunciati che hanno valore di verità TRUE in tutte le situazioni (tautologie), gli enunciati che hanno valore di verità FALSE in tutte le situazioni (contraddizioni) e gli enunciati il cui valore di verità varia a seconda delle situazioni (enunciati contingenti).
Abbiamo così potuto evidenziare l'interessante relazione che intercorre fra un'argomentazione e l'enunciato condizionale ad essa corrispondente (il quale ha per antecedente la congiunzione delle premesse e per conseguente la conclusione): un'argomentazione è valida se e solo se il condizionale ad essa corrispondente è una tautologia.
Due enunciati si dicono logicamente equivalenti se hanno gli stessi valori di verità nelle stesse situazioni. Perciò due tautologie sono sempre logicamente equivalenti, e così pure due contraddizioni. Altre due forme importanti di equivalenza logica sono le leggi di De Morgan e l'esportazione.
Una terza equivalenza logica di rilievo è quella che intercorre fra un enunciato qualsiasi e la sua forma normale congiuntiva (FNC), costituita da una congiunzione di disgiunzioni. Abbiamo definito due algoritmi per determinare un enunciato in FNC che sia logicamente equivalente a un enunciato dato, non espresso in FNC.
Abbiamo poi applicato questi concetti all'analisi di insiemi di enunciati. Se esiste una situazione in cui tutti gli enunciati di un dato insieme abbiano valore di verità TRUE, tali enunciati si dicono simultaneamente soddisfacibili. Se gli enunciati di un insieme sono usati come premesse di un'argomentazione, l'insieme si dice coerente se non esiste nessuna conclusione valida che sia una contraddizione; altrimenti l'insieme si dice contradditorio.
Abbiamo infine descritto la notazione polacca, che rappresenta i connettivi con lettere maiuscole corsive ed esprime gli enunciati molecolari in forma prefissa. La notazione polacca rende superfluo l'uso delle parentesi e semplifica alcune operazioni sugli enunciati, come l'identificazione del connettivo principale.

7.7 ESERCIZI

A. Determinare se ciascuno degli enunciati seguenti è tautologico, contraddittorio o contingente:

1. $(((A \to B) \wedge A) \to B)$
2. $(((A \to B) \wedge A) \to \neg B)$
3. $(((A \to B) \wedge \neg B) \to \neg A)$
4. $(((A \to B) \wedge \neg B) \to A)$
5. $(((A \to B) \wedge \neg A) \to \neg B)$
6. $(((A \to B) \wedge B) \to A)$
7. $(((A \vee B) \wedge \neg A) \to B)$
8. $(((A \vee B) \wedge \neg A) \to \neg B)$
9. $(((A \to B) \wedge (B \to C)) \to (A \to C))$
10. $(((A \to B) \wedge (A \to C)) \to (B \to C))$
11. $((A \wedge \neg A) \to B)$
12. $((A \wedge \neg A) \to B \neg)$
13. $(((A \wedge \neg B) \wedge A) \wedge \neg B)$
14. $((A \vee B) \wedge (\neg A \wedge \neg B))$
15. $(A \wedge (\neg A \wedge B))$
16. $(A \wedge (\neg A \wedge \neg B))$
17. $(A \leftrightarrow A)$
18. $(A \leftrightarrow \neg A)$
19. $(((A \leftrightarrow B) \wedge A) \leftrightarrow B)$
20. ((A NAND A)∨A)
21. ((A NAND B) NAND (A NAND B))
22. ((A NOR A) NOR (B NOR B)
23. (A NAND (B NAND B))
24. ((A NOR A)∧A)
25. ((A∧(A NAND (B NAND C))) → C)
26. ((A∧ (A NAND (B NAND B))) → B)
27. ((A NAND B) NOR A)
28. ((A NOR B) NAND A)
29. (A XOR A)
30. ((A XOR B)∧¬A)
31. (((A XOR B)∧¬A) →B)
32. ¬(A XOR ¬A)

B. Costruire la tavola di verità di

$$((A \to B) \leftrightarrow (\neg B \to \neg A))$$

e dimostrare che si tratta di una tautologia.

C. Condizionale corrispondente a un'argomentazione

1. Quali sono i condizionali corrispondenti alle argomentazioni seguenti?

a.
1. A
2. B
3. C
4. D

∴5. E

b.
1. (A∨B)
2. C

∴3. D

c. 1. A
   2. B
   3. C
   ∴4. $\neg(D \wedge C)$

d. 1. $(A \wedge B)$
   2. C
   ∴3. D

2. Dire se le argomentazioni seguenti sono valide o no, determinando se i condizionali ad esse corrispondenti sono tautologie.
   a. Se Paola va allo zoo, vede un elefante. Oggi Paola ha visto un elefante, quindi deve essere stata allo zoo.
   b. Se Paola va allo zoo, vede un elefante. Oggi Paola è andata allo zoo, quindi deve aver visto un elefante.
3. Quali sono le argomentazioni corrispondenti ai seguenti enunciati condizionali?
   a. $(A \rightarrow B)$
   b. $(((A \wedge B) \wedge C) \rightarrow D)$
   c. $((A \wedge (B \wedge C)) \rightarrow D)$
   d. $(((A \vee B) \wedge C) \rightarrow D)$
   e. $((A \vee (B \wedge C)) \rightarrow D)$

D. Dire quali delle seguenti coppie di enunciati sono logicamente equivalenti:
   1. $(\neg A \vee B)$, $(A \rightarrow B)$
   2. $\neg(A \wedge \neg B)$, $(A \rightarrow B)$
   3. A, $(A \wedge A)$
   4. A, $(A \vee A)$
   5. $(A \wedge (B \wedge C))$, $((A \wedge B) \wedge C)$
   6. $(A \vee (B \vee C))$, $((A \vee B) \vee C)$
   7. $\neg(A \rightarrow B)$, $(\neg A \wedge \neg B)$
   8. $\neg(A \wedge B)$, $(\neg A \wedge \neg B)$
   9. $\neg(A \vee B)$, $(\neg A \vee \neg B)$
   10. $(A \rightarrow B)$, $(B \rightarrow A)$
   11. $(A \leftrightarrow B)$, $((A \rightarrow B) \wedge (B \rightarrow A))$
   12. $(A \leftrightarrow B)$, $((A \wedge B) \vee (\neg A \wedge \neg B))$
   13. $(A \vee (B \wedge C))$,$((A \vee B) \wedge (A \vee C))$
   14. $(A \wedge (B \vee C))$, $((A \wedge B) \vee (A \wedge C))$
   15. $\neg\neg A$, $\neg\neg\neg\neg A$
   16. $((A \vee \neg A) \wedge C)$, $((B \vee \neg B) \wedge C)$
   17. $((A \vee \neg A) \wedge C)$, C
   18. $((A \wedge \neg A) \vee C)$, C
   19. (A XOR B), $\neg(A \leftrightarrow B)$
   20. $\neg(A \leftrightarrow B)$, $(A \leftrightarrow \neg B)$
   21. (A XOR B), $((A \vee B) \wedge \neg(A \wedge B))$
   22. (A NAND A) (A NOR A)
   23. ((A NAND A) NAND (B NAND B)), $(A \wedge B)$
   24. $(A \rightarrow B)$, (((A NOR A) NOR B) NOR ((A NOR A) NOR B))
   25. ((A NOR B) NOR (A NOR B)), $(A \vee B)$

E. Dato un enunciato **P**, costruire un altro enunciato **Q** nel modo seguente:
 1. Sostituire ogni (**S**→**T**) di **P** con (¬**S**∨**T**) e ogni (**S**↔**T**) con ((¬**S**∨**T**) ∧ (¬**T**∨**S**)).
 2. Sostituire tutti gli enunciati atomici di **P** con le loro negazioni.
 3. Sostituire ogni ∨ con ∧ e ogni ∧ con ∨.
 4. Negare il risultato.
 5. Eliminare tutte le doppie negazioni (cioè sostituire ogni ¬ ¬**R** con **R**).

 Ad esempio, se **P** = '¬(¬A∧B)', eseguendo i passi da 1 a 4 si ottiene '¬ ¬(¬ ¬A∨¬B)'; eseguendo poi il passo 5 si ha **Q** = '(A∨¬B)'. Per ogni enunciato **P**, l'enunciato **Q** costruito nel modo sopra descritto si dice *duale* di **P**.
 1. Costruire il duale di ciascuno degli enunciati seguenti:
    a. ¬(A∧B)
    b. (A∨B)
    c. ((A∨B) →C)
    d. (¬A→(B∧C))
 2. Dimostrare che ogni enunciato **P** è logicamente equivalente al suo duale.

F. Usando l'algoritmo FNC-1 o FNC-2, trovare degli enunciati in forma normale congiuntiva che siano logicamente equivalenti a:
 1. (A∨A)
 2. (A∧A)
 3. (A→B)
 4. ((A∧B)∨(A∧C))
 5. ((A∨B)∧(A∧C))
 6. ((A∧(A→B))→B)
 7. ((A∧B)∨(A∧C)∨(A∧B∧¬C))
 8. ((A∨B)∧(A∨C)∧(A∨B∨¬C))
 9. ((A∧B)∧(¬A∨¬B))
 10. ((¬A∧B)∧(A∨¬B))
 11. ((¬A∧B)∨(A∧¬B))

G. Forma normale disgiuntiva
 1. Definire la forma normale disgiuntiva (FND).
    (*Suggerimento*: come un enunciato in FNC è una congiunzione di disgiunzioni, così un enunciato in FND sarà una disgiunzione di congiunzioni).
 2. Scrivere un algoritmo che determini un enunciato equivalente in FND per ogni enunciato dato.
 3. Convertire in FND gli enunciati seguenti:
    a. (A∧A)
    b. (A→B)
    c. ((A∨B) → C)
    d. (A∧¬(B∨C))
    e. (A→ (B∧C))

H. Notazione polacca
 1. Tradurre gli enunciati seguenti da notazione infissa a notazione polacca:
    a. (((D→B)∧D) →B)
    b. (((D→B)∧(B→E)) →(D→F))

c. $((D \to B) \leftrightarrow (\neg B \to \neg D))$
   d. $(D \wedge (B \wedge (F \wedge G)))$
   e. $\neg (D \wedge (B \vee (\neg F \wedge G)))$
   f. $(((D \wedge B) \wedge F) \wedge G)$
2. Tradurre gli enunciati seguenti da notazione polacca a notazione infissa:
   a. *A*B*C*D*E*FG
   b. *CKC*BDBD
   c. *C*B*A*BD
   d. *CK*BD*K*BD
   e. *ENN*BB
   f. *E*B*NN*B
3. Scrivere un algoritmo che effettui la conversione da notazione polacca a notazione infissa.
4. Una notazione postfissa, detta notazione polacca inversa (NPI) è definita nel modo seguente:
   I. Ogni enunciato atomico è ben formato in NPI.
   II. Se **P** o **Q** sono entrambi ben formati in NPI, allora lo sono anche
   P*N*
   PQ*K*
   PQ*A*
   PQ*C*
   PQ*E*
   III. Nient'altro è ben formato in NPI.
   In questa notazione
   P*N* $= \neg \mathbf{P}$
   PQ*K* $= (\mathbf{P} \wedge \mathbf{Q})$
   PQ*A* $= (\mathbf{P} \vee \mathbf{Q})$
   PQ*C* $= (\mathbf{P} \to \mathbf{Q})$
   PQ*E* $= (\mathbf{P} \leftrightarrow \mathbf{Q})$
   a. Scrivere un algoritmo per tradurre da notazione infissa a NPI.
   b. Scrivere un algoritmo per tradurre da NPI a notazione infissa.
   c. Applicare agli enunciati dell'Esercizio H.1 l'algoritmo per tradurre da notazione infissa a NPI.

8

Logica enunciativa: un sistema di deduzione naturale

I metodi per determinare se un'argomentazione è valida sono completi e realizzabili, come dimostrano gli algoritmi: possono essere usati per determinare, in un tempo finito, se un'argomentazione espressa in forma simbolica nella logica enunciativa è valida oppure no.

Si potrebbero però avanzare alcune riserve a proposito di questi metodi. Innanzitutto essi richiedono che ci si occupi, in termini molto espliciti, dei valori di verità degli enunciati: si deve calcolare quando tutti gli enunciati rilevanti hanno valore di verità TRUE in diverse situazioni, e si ottiene come risultato una tabella che illustra quando certi enunciati valgono TRUE e quando valgono FALSE. Inoltre con questi metodi è particolarmente difficile dimostrare che un'argomentazione è valida, in quanto l'applicazione del metodo delle tavole di verità ha durata massima quando l'argomentazione è valida, mentre ha una durata minore se l'argomentazione non è valida. Infine, non è plausibile che qualcuno, per ragionare, usi esplicitamente le complicate relazioni fra verità e falsità che abbiamo studiato. Sarebbe comodo disporre di un metodo più semplice per ragionare e per verificare la validità di un ragionamento; ad esempio, un metodo che richieda meno calcoli e meno passaggi scritti intermedi.

Per queste, e per altre ragioni storiche, la maggior parte dei logici, dei matematici e degli informatici nelle applicazioni pratiche preferisce ricorrere a un *sistema formale di deduzione* per svolgere i ragionamenti e dimostrarne la validità. Un sistema formale di deduzione ha lo scopo fondamentale di dimostrare la validità di un'argomentazione, mentre generalmente non viene usato per dimostrare che un'argomentazione non è valida.

Un sistema formale di deduzione dimostra la validità di un'argomentazione senza ricorrere ai concetti di TRUE e FALSE, in quanto si occupa della forma, o modello, di un'argomentazione, senza considerare, neppure temporaneamente, i valori di verità degli enunciati che vi compaiono, o le situazioni del mondo reale cui tali enunciati fanno riferimento. Questo sistema di deduzione si dice formale appunto perché si occupa solo della *forma* di un'argomentazione, e non del suo contenuto.

Lo strumento principale di un sistema formale è la *derivazione*, detta anche *deduzione* o *prova*. Una derivazione dimostra che certe stringhe di simboli seguono da altre stringhe secondo regole prefissate. In un sistema formale di deduzione, una derivazione è una deduzione di conseguenze logiche a partire da determinate premesse; tale deduzione viene effettuata senza considerare né il significato dei termini degli enunciati, né i valori di verità degli enunciati. Una derivazione si occupa solo del problema di determinare se un enunciato, considerato come una struttura simbolica, segue da altri enunciati secondo regole prefissate. Nella logica deduttiva, tali regole sono dette *regole di inferenza*, perché hanno lo scopo di trasformare determinati enunciati in altri che si possano inferire in modo valido. Le regole di un sistema formale possono essere viste come modi prestabiliti per trasformare alcune stringhe in altre.
Poiché un sistema formale ha delle regole che devono essere seguite, una derivazione è paragonabile a un *gioco*. I pezzi e le mosse di un gioco di solito sono privi di significato al di fuori del gioco stesso, ma il gioco deve seguire determinate regole. Analogamente, i pezzi del gioco "derivazione" non hanno un significato preciso (o meglio, non occorre considerarne il significato quando si costruisce la derivazione), ma devono essere manipolati secondo regole prestabilite.

8.1 UN SEMPLICE SISTEMA FORMALE: IL GIOCO DELLE STELLE E DELLE BARRE

Per cominciare, consideriamo un paio di giochi che usano quattro simboli: le lettere 'A' e 'B', le stelle e le barre:

A B ☆ /

Questi simboli possono essere combinati in molti modi, utilizzando un numero arbitrario di copie di ciascun simbolo. Ad esempio,

AB☆/
///☆AA
/A☆B/

sono tutte stringhe costruite con questi simboli. Stabiliamo che alcune delle stringhe possibili siano vincenti e le altre siano perdenti. Un gioco potrebbe consistere nel costruire il maggior numero possibile di stringhe vincenti (il gioco potrebbe essere reso più interessante introducendo dei vincoli, come, ad esempio, dei limiti di tempo, ma tali estensioni non saranno qui prese in considerazione). Un altro gioco potrebbe consistere nel decidere se una stringa data, ad esempio

S /////A☆B/☆A/☆A/☆B/☆B//

è vincente o perdente.

Ovviamente, come in qualsiasi altro gioco, devono esserci delle regole; nel nostro caso, servono regole che specifichino quali stringhe siano vincenti e quali perdenti. Sia per il gioco di costruzione, sia per quello di decisione, adotteremo le tre regole seguenti:

1. Il simbolo
   A
   da solo è una stringa vincente.
2. Il simbolo
   B
   da solo è una stringa vincente.
3. Se S_1 e S_2 sono stringhe vincenti, allora
   $/S_1 \star S_2/$
   è una stringa vincente. In altre parole, per costruire una stringa vincente si collegano due stringhe vincenti mediante il simbolo '☆' e si racchiude il risultato fra due barre.

È sottinteso che nessun'altra stringa, al di fuori di quelle ottenute con queste tre regole, è vincente.
Per chiarire il significato di queste regole, cominciamo a giocare al gioco di costruzione. In base alle prime due regole, la stringa

A

e la stringa

B

sono vincenti. Ora che abbiamo due stringhe vincenti, possiamo applicare la regola 3; assumendo S_1 = 'A' e S_2 = 'B', otteniamo la stringa vincente

/A☆B/

Applichiamo nuovamente la regola 3, assumendo stavolta S_1 = 'B' e S_2 = 'A'; otteniamo così la stringa vincente

/B☆A/

Abbiamo ora a disposizione quattro stringhe vincenti; è chiaro che possiamo ottenerne molte altre, continuando ad applicare la terza regola. Ad esempio, posto S_1 = 'A' e S_2 = '/B☆A/', otteniamo la stringa vincente

/A☆/B☆A//

La stringa

/A☆/B☆A/

è invece perdente, perché non può essere ottenuta mediante le tre regole viste. Come si può dimostrare che la stringa '/A☆/B☆A/' è perdente? Purtroppo non basta tentare di produrla senza riuscirci, perché, soprattutto nel caso di stringhe più lunghe e complesse, il fallimento può sempre avere due cause: la stringa è effettivamente perdente, oppure è vincente, ma non si è riusciti a produrla per mancanza di abilità o di impegno.
Per dimostrare che '/A☆/B☆A/' è una stringa perdente, si osservi che tutte le stringhe vincenti o sono costituite dai simboli 'A' o 'B' isolati, o hanno la seguente struttura:

1. Il primo e l'ultimo simbolo sono barre.
2. Rimuovendo il primo e l'ultimo simbolo, si ottiene una stella compresa fra due stringhe vincenti.

La stringa che stiamo considerando non soddisfa la seconda condizione; rimuovendo infatti la coppia esterna di barre si ottiene

A☆/B☆A

Questa stringa non può essere ottenuta unendo due stringhe vincenti mediante una stella; infatti violano la prima condizione sia la stringa a destra della prima stella, cioè

/B☆A

sia la stringa a sinistra della seconda stella, cioè

A☆/B

Poiché la stringa non si può analizzare in altro modo, deve essere perdente; non è infatti possibile analizzarla in modo da dimostrare che essa sia costituita da due stringhe vincenti unite da una stella.

8.2 SISTEMI FORMALI

Il gioco delle stelle e delle barre evidenzia le componenti fondamentali del tipo più semplice di sistema formale. Un *sistema formale* è costituito da tre elementi caratteristici:

1. Un insieme di simboli con cui formare stringhe e un insieme di regole che specificano che cos'è una stringa ben formata. Nel nostro gioco, i simboli sono 'A', 'B', ☆ e /; ogni stringa si considera ben formata, purché sia costituita da questi simboli.
2. Un elenco di stringhe vincenti, detto insieme degli *assiomi*. Nel nostro gioco, gli assiomi sono definiti dalle regole 1 e 2.
3. Un insieme di metodi per costruire nuove stringhe vincenti, detto insieme delle *regole di inferenza*. Il nostro gioco ha una sola regola di questo tipo: la 3.

Una stringa vincente è una particolare stringa ben formata, costituita da un esemplare di assioma oppure costruita, a partire da stringhe vincenti già note, mediante applicazioni delle regole di inferenza.
In logica e in matematica le stringhe vincenti sono dette *teoremi*. Nel nostro gioco,

/A☆B/

è un teorema; è infatti una stringa vincente, in virtù delle regole 1 e 2 e di una applicazione della regola 3.
Il sistema formale di deduzione che svilupperemo nei prossimi paragrafi sarà piuttosto simile al gioco delle stelle e delle barre, almeno nei suoi tratti fondamentali. Avrà stringhe ben formate e regole per generare nuove stringhe vincenti; sarà sottinteso che sono vincenti solo le stringhe ottenute applicando tali regole. Non esisteranno però assiomi, cioè non saranno definite stringhe automaticamente vincenti; un sistema di deduzione privo di assiomi è detto *sistema di deduzione naturale*.

8.3 UN SISTEMA DI DEDUZIONE NATURALE

Il gioco delle stelle e delle barre è un buon esempio di sistema formale, in quanto vi sono regole rigorose per generare stringhe vincenti ed esistono procedimenti per costruire derivazioni usando tali regole; non è però un sistema formale *di deduzione*. Un sistema formale di deduzione, infatti, gode di due proprietà aggiuntive: in primo luogo, tutte le stringhe di un sistema formale di deduzione sono enunciati; in secondo luogo, le regole con cui si derivano enunciati da altri enunciati non sono arbitrarie; una regola di inferenza deve infatti permettere di derivare solo gli enunciati che seguono logicamente da enunciati precedenti. In più, un sistema di deduzione *naturale* è privo di assiomi.
Una regola di un sistema di deduzione deve avere la caratteristica di *conservare la verità*, cioè di non far mai passare da enunciati veri a enunciati falsi. La conservazione della verità è la preoccupazione principale della logica, come abbiamo visto nel Capitolo 1, ed è anche il fondamento del concetto di validità. Una buona argomentazione, infatti, è innanzitutto un'argomentazione che conserva la verità:

se le premesse sono tutte vere, deve essere vera anche la conclusione.
Tutte le regole di un sistema formale di deduzione devono dunque conservare la verità, nel senso che non deve essere possibile partire da enunciati veri e giungere ad enunciati falsi applicando correttamente le regole; sono invece ammesse regole che possano far passare da enunciati falsi ad enunciati veri: una macchina che trasformi oro in fango è deprecabile, ma una che trasformi fango in oro è senz'altro ben accetta!
Come abbiamo detto, e come abbiamo visto nel gioco delle stelle e delle barre, l'applicazione più importante di un sistema formale consiste nel produrre una derivazione, cioè una sequenza di passi tutti leciti in base alle regole del sistema formale. Per facilitare la lettura e la verifica di una derivazione, numereremo ogni passo e ne forniremo una giustificazione.

8.4 INTRODUZIONE DELLA CONGIUNZIONE

Consideriamo la prima regola del sistema formale di deduzione per la logica enunciativa: l'*introduzione della congiunzione*, che abbrevieremo con ∧INTR:

∧INTR Se **P** e **Q** sono due enunciati già comparsi nella derivazione, si può derivare la loro congiunzione:
(P∧Q)

In altre parole, se due enunciati **P** e **Q** sono già stati ottenuti nella derivazione in base alle regole, li si può, per così dire, mettere insieme, unendoli con il connettivo di congiunzione.
La maggior parte delle deduzioni ha lo scopo di dimostrare che un enunciato (la conclusione) segue da altri enunciati in base alle regole di inferenza ammesse. In tal caso le premesse sono dei dati, quindi si possono accettare come enunciati leciti, senza preoccuparsi del fatto che non sono giustificate in altro modo.
Una deduzione, dunque, comincia tipicamente con enunciati che hanno come unica giustificazione il fatto di essere le premesse dell'argomentzione data; come giustificazione di tali enunciati in una deduzione, scriveremo semplicemente PREMESSA. Ad esempio, se un'argomentazione ha le premesse

A
(B→C)

l'inizio di una deduzione che usi queste premesse sarà:

1. A : PREMESSA
2. (B→C) : PREMESSA

Ma come può proseguire la deduzione? A parte l'elencazione delle premesse del-

l'argomentazione, che abbiamo già completato, abbiamo per ora a disposizione solo la regola $\land$INTR, che permette di combinare due enunciati precedenti qualsiasi mediante il connettivo $\land$. Notiamo che 'A' e '$(B\to C)$' sono due enunciati precedenti, il primo corrispondente alla **P** della regola e il secondo corrispondente alla **Q**. Possiamo quindi continuare la deduzione:

1. A	: PREMESSA
2. $(B\to C)$	: PREMESSA
3. $(A\land(B\to C))$	: $\land$INTR applicata alle righe 1 e 2

La regola $\land$INTR può poi essere applicata piu volte:

1. A	: PREMESSA
2. $(B\to C)$	: PREMESSA
3. $(A\land(B\to C))$	: $\land$INTR,1,2
4. $((B\to C)\land A)$	: $\land$INTR,2,1
5. $(A\land A)$	: $\land$INTR,1,1
6. $((A\land A)\land(A\land(B\to C)))$	: $\land$INTR,5,3

È chiaro che si potrebbe continuare indefinitamente, applicando $\land$INTR a due enunciati precedenti qualsiasi. Si noti inoltre che le due righe precedenti cui si applica la regola $\land$INTR possono anche essere coincidenti; nella deduzione sopra riportata abbiamo infatti:

1. A	: PREMESSA
.	
.	
.	
5. $(A\land A)$	: $\land$INTR,1,1

La caratteristica più importante cui si deve mirare nella formulazione delle regole del sistema formale di deduzione è la conservazione del valore di verità. La regola $\land$INTR conserva il valore di verità? In altre parole, se tutti gli enunciati precedenti in una derivazione hanno valore di verità TRUE, la regola $\land$INTR permette di derivare solo enunciati aventi valore di verità TRUE?
Per rendersi conto che $\land$INTR conserva il valore di verità si può confrontare il modello degli enunciati in un'applicazione di $\land$INTR con la tavola di verità del connettivo $\land$.

1. **P**	: PREMESSA
2. **Q**	: PREMESSA
3. **(P$\land$Q)**	: $\land$INTR,1,2

È possibile che **(P$\land$Q)** sia falso se **P** e **Q** sono entrambi veri? Ovviamente no, come abbiamo visto nel Capitolo 3; esaminiamo infatti la tavola di verità:

V(**P**)	V(**Q**)	V(**P**∧**Q**)
0	0	0
0	1	0
1	0	0
1	1	1

Non esiste nessuna situazione in cui **P** e **Q** valgano TRUE e (**P**∧**Q**) valga FALSE: dunque la regola ∧INTRO conserva il valore di verità.
Se una regola come ∧INTR conserva il valore di verità, la si può applicare più e più volte ad enunciati veri, senza temere di poter in tal modo introdurre un enunciato falso: se ∧INTR conserva il valore di verità una volta, lo conserverà anche una seconda, una terza, e così via. Se le premesse sono tutte vere, l'applicazione di regole che conservano il valore di verità, ripetuta un numero qualsiasi di volte e in ordine qualsiasi, non farà mai ottenere un enunciato falso. Questo fatto ci permette di effettuare le derivazioni in tutta sicurezza: se una derivazione usa solo regole che conservano il valore di verità, allora tale derivazione nel suo complesso conserverà il valore di verità, ossia non permetterà mai di passare da premesse vere a una conclusione falsa.

8.5 IL FORMATO DI UNA DERIVAZIONE

Tutte le derivazioni saranno sempre presentate in un formato prefissato: ogni riga è numerata e contiene un enunciato seguito dalla spiegazione del motivo per cui è lecito scrivere tale enunciato; questa spiegazione è detta *giustificazione* dell'enunciato.
Finora abbiamo a disposizione solo due giustificazioni per scrivere un enunciato in una derivazione:

1. L'enunciato è una delle premesse dell'argomentazione, e quindi ha la giustificazione PREMESSA.
2. L'enunciato è stato derivato dagli enunciati di due righe precedenti mediante la regola ∧INTR.

Più in generale, ogni riga di una derivazione avrà le seguenti caratteristiche:

1. La riga comincia con un numero intero positivo, detto numero di riga, seguito da un punto. Non possono esistere due righe distinte di una derivazione che abbiano lo stesso numero di riga. Fra breve modificheremo leggermente queste specifiche, ammettendo che il numero di riga possa essere preceduto da certi simboli.
2. Nella riga compare poi un enunciato, cioè una stringa ben formata in base alle regole esposte nei Capitoli dal 3 al 5.

3. Al termine della riga ci sono due punti (:), seguiti dalla giustificazione dell'enunciato. Tale giustificazione può essere o PREMESSA, o il nome della regola del sistema di deduzione in base alla quale è lecito scrivere l'enunciato. Se la giustificazione riguarda l'applicazione di una regola a righe precedenti della derivazione, occorre specificare anche i numeri di tali righe. La giustificazione di una riga può dunque essere:
   : PREMESSA
   che non richiede riferimenti a righe precedenti della derivazione, oppure, ad esempio,
   : ∧INTR,1,2
   che fa riferimento alle righe 1 e 2.

8.6 ELIMINAZIONE DELLA CONGIUNZIONE

Introduciamo ora l'altra regola relativa al connettivo ∧: l'*eliminazione della congiunzione*, o ∧ELIM:

∧ELIM   Se (**P**∧**Q**) è un enunciato precedente della derivazione, si può derivare o il primo congiunto:
  P
o il secondo congiunto:
  Q

La breve derivazione seguente illustra un'applicazione corretta di questa nuova regola.

1.	(A∧B)	: PREMESSA
2.	C	: PREMESSA
3.	A	: ∧ELIM,1

Qui 'A' svolge il ruolo di **P**, mentre 'B' svolge il ruolo di **Q**. A volte l'enunciato derivato può essere più complesso; si consideri ad esempio questa derivazione:

1.	(D∧(E→F))	: PREMESSA
2.	E	: PREMESSA
3.	(E→F)	: ∧ELIM,1

Qui l'enunciato 'D' svolge il ruolo di **P** nella regola, mentre l'enunciato molecolare '(E→F)' svolge il ruolo di **Q**. In altre parole, la regola ∧ELIM afferma che da una qualsiasi congiunzione

(<enunciato 1> ∧ <enunciato 2>)

si può derivare

<enunciato 1>

da solo, oppure, a scelta,

<enunciato 2>

da solo. Questi enunciati derivati possono essere atomici o molecolari; l'unica condizione richiesta è che l'enunciato originario sia una congiunzione, cioè abbia ∧ come connettivo principale.
È facile verificare che ∧ELIM conserva il valore di verità: se

(P∧Q)

ha valore di verità TRUE, allora anche

P

ha valore di verità TRUE, così come

Q

Basta consultare le tavole di verità del Capitolo 3 per rendersene conto. Dunque la regola ∧ELIM, come ∧INTR, non farà mai passare da enunciati veri a enunciati falsi.
Si osservi che la regola ∧INTR converte due enunciati precedenti in un enunciato nuovo, mentre la regola ∧ELIM converte un enunciato in un altro enunciato; infatti ∧ELIM permette di derivare sia l'uno sia l'altro congiunto, ma in ogni applicazione se ne può derivare uno solo. Ciò significa che la giustificazione di un enunciato ottenuto con la regola ∧ELIM deve far riferimento a un solo numero di riga precedente, mentre un enunciato giustificato da ∧INTR ne deve citare due.
Vediamo alcuni esempi di giustificazione:

: ∧ELIM,3
: ∧INTR,2,4
: ∧INTR,1,1
: ∧ELIM,7

Le seguenti giustificazioni sono invece sempre scorrette:

: ∧ELIM,1,2	[scorretta, perché ∧ELIM deve far riferimento a una e una sola riga precedente]
: ∧INTR,3	[scorretta, perché ∧INTR deve far riferimento a due righe precedenti]

: PREMESSA,3 [scorretta, perché quando la giustificazione è PREMESSA non si deve citare nessuna riga precedente]

Le regole ∧INTR e ∧ELIM presentano una certa simmetria, come i nomi stessi suggeriscono: ∧INTR introduce una nuova congiunzione, cioè un enunciato avente ∧ come connettivo principale, mentre ∧ELIM fa passare da una congiunzione a un enunciato che contiene un simbolo di congiunzione in meno.
I nomi di queste due regole forniscono qualche indicazione sui casi in cui conviene applicarle: ∧INTR verrà applicata quando si voglia ottenere un enunciato contenente il connettivo ∧, mentre ∧ELIM verrà applicata quando si voglia eliminare un connettivo ∧ o ridurre il numero di ∧ negli enunciati precedenti. La maggior parte delle regole, in effetti, presenterà questa simmetria, permettendo o l'introduzione o l'eliminazione di un connettivo. Ogni regola fornirà perciò qualche indicazione sui casi in cui conviene applicarla.

8.7 DIMOSTRAZIONE DELLA VALIDITÀ DI UNA ARGOMENTAZIONE MEDIANTE UNA DERIVAZIONE

Siamo ora in grado di mettere in relazione una derivazione con un'argomentazione. La derivazione della conclusione di un'argomentazione a partire dalle sue premesse ha le seguenti caratteristiche:

1. Ogni enunciato di ogni riga è giustificato (*a*) dal fatto di essere una premessa dell'argomentazione data, oppure (*b*) dal fatto di essere stato correttamente derivato da righe precedenti mediante l'applicazione di regole permesse.
2. L'enunciato dell'ultima riga della derivazione è la conclusione dell'argomentazione data.

Ad esempio, se è data l'argomentazione valida

A
B
∴ (A∧B)

una derivazione corretta della conclusione a partire dalle premesse è:

1. A	: PREMESSA
2. B	: PREMESSA
3. (A∧B)	: ∧INTR,1,2

L'ultima riga di questa derivazione è la conclusione dell'argomentazione data, e

le uniche giustificazioni sono PREMESSA e una delle regole ammesse. Parleremo in questi casi di derivazione della conclusione, o di dimostrazione della validità di un'argomentazione.
Se è data l'argomentazione valida

(C∧D)
∴ C

una derivazione corretta della conclusione dalle premesse è:

1. (C∧D) : PREMESSA
2. C : ∧ELIM,1

Per la prima argomentazione [A, B, ∴, (A∧B)], è facile rendersi conto che si deve usare ∧INTR, giacché la conclusione contiene il connettivo ∧, ma le premesse no. Per la seconda argomentazione [(C∧D), ∴ C], è chiaro che si deve usare ∧ELIM, perché la premessa contiene un connettivo ∧, ma la conclusione no.
Consideriamo ora diversi altri esempi di derivazione.

Argomentazione 1
(A∧B)
∴ (B∧A)

Derivazione
1. (A∧B) : PREMESSA
2. A : ∧ELIM,1
3. B : ∧ELIM,1
4. (B∧A) : ∧INTR,3,2

Si noti bene che la corretta applicazione di ∧INTR alla riga 4 richiede che si citi la riga da cui si ricava il primo congiunto 'B' (riga 3) e la riga da cui si ricava il secondo congiunto 'A' (riga 2).

Argomentazione 2
A
∴ A

Derivazione
1. A : PREMESSA

È evidentemente una prova strana, però è conforme ai requisiti di una derivazione corretta: abbiamo usato solo la giustificazione PREMESSA, e l'ultima riga, che coincide con la prima, è la conclusione.

Argomentazione 3

$(A \wedge B)$
C
$\therefore (C \wedge A)$

Derivazione

1. $(A \wedge B)$: PREMESSA
2. C : PREMESSA
3. A : $\wedge$ELIM,1
4. $(C \wedge A)$: $\wedge$INTR,2,3

Argomentazione 4

$((A \wedge B) \wedge C)$
$\therefore (A \wedge C)$

Derivazione

1. $((A \wedge B) \wedge C)$: PREMESSA
2. $(A \wedge B)$: $\wedge$ELIM,1
3. A : $\wedge$ELIM,2
4. C : $\wedge$ELIM,1
5. $(A \wedge C)$: $\wedge$INTR,3,4

Argomentazione 5

$(A \wedge (C \rightarrow D))$
$(\neg A \vee B)$
$\therefore (C \rightarrow D)$

Derivazione

1. $(A \wedge (C \rightarrow D))$: PREMESSA
2. $(\neg A \vee B)$: PREMESSA
3. $(C \rightarrow D)$: $\wedge$ELIM,1

Prima di introdurre la prossima coppia di regole, consideriamo che cosa illustra una derivazione. Una derivazione mostra che un enunciato, cioè l'ultima riga della derivazione stessa, può essere derivato da enunciati precedenti mediante l'applicazione di regole di inferenza. Si sa che queste regole godono della proprietà di conservare il valore di verità; siamo perciò certi che se gli enunciati introdotti con l'unica giustificazione diversa da una regola, cioè PREMESSA, sono tutti veri, allora anche l'ultima riga della derivazione sarà vera. Ma in tal caso l'argomentazione è valida; dunque una derivazione corretta della conclusione dalle premesse dimostra che l'argomentazione è valida. Nella maggior parte dei casi, questo modo di dimostrare la validità di un'argomentazione è molto più semplice della costruzione di una tavola di verità. Una derivazione inoltre, rispetto a un'ingombrante tavola di verità, permette anche di capire meglio *perché* un'argomentazione è valida.

8.8 SOTTOPROVE E INTRODUZIONE DELLA NEGAZIONE

Si consideri l'argomentazione seguente.

Non posso guardare la TV stasera e, allo stesso tempo, finire di fare i compiti. Stasera guarderò la TV. Perciò non finirò di fare i compiti.

Questa argomentazione può essere espressa simbolicamente così:

Argomentazione 6
$\neg(G \wedge C)$
G
$\therefore\ \neg C$

dove

G = 'Stasera guarderò la TV.'
C = 'Finirò di fare i compiti.'

Un ragionamento a sostegno della validità di questa argomentazione potrebbe essere il seguente:

Non posso guardare la TV stasera e, allo stesso tempo, finire di fare i compiti.
Stasera guarderò la TV.
Supponiamo però che io riesca anche a finire i compiti.
Ciò significherebbe che stasera guarderei la TV *e*, in qualche modo, finirei anche i compiti.
Ciò contraddice però la prima affermazione, secondo la quale io non posso fare entrambe le cose.
La supposizione che io riesca a finire i compiti deve perciò essere sbagliata.
Dunque, non finirò i compiti.

Rappresentando gli enunciati in forma simbolica, potremmo esprimere questo ragionamento nel modo seguente:

$\neg(G \wedge C)$: PREMESSA
G : PREMESSA
Supponiamo C.
Allora $(G \wedge C)$.
Ma la prima premessa è $\neg(G \wedge C)$.
Dunque la supposizione C deve essere sbagliata.
Perciò $\neg C$.

I passi che vanno da "Supponiamo C" a "Perciò ¬C" hanno l'effetto di introdurre un segno di negazione. Essi illustrano il tipo di ragionamento che sta alla base di una nuova regola, l'*introduzione della negazione*, o ¬INTR. Questo ragionamento, che non è espresso esattamente nella forma richiesta per le derivazioni, contiene due nuove idee importanti. Una prima idea consiste nel supporre qualcosa. Una supposizione come 'Finirò di fare i compiti' non è una PREMESSA: non è l'argomentazione assegnata che ci impone di considerarla vera. Non è però neppure derivabile da righe precedenti in base a regole come ∧ELIM o ∧INTR. Abbiamo invece introdotto questo enunciato *come se* esso fosse logico, per vedere che cosa ne consegue. Chiameremo *assunzioni* queste supposizioni. Poiché le assunzioni sono dei tentativi, non forniti come PREMESSE, né derivabili da righe precedenti, devono essere usate con molta cura.

SOTTOPROVE

Una *sottoprova* è una digressione dalla prova principale; più in generale, è una prova all'interno di un'altra prova. La supposizione con cui comincia una sottoprova può essere considerata come una specie di premessa temporanea: l'assunzione è una "premessa" solo per la durata della sottoprova, ma non per l'intera derivazione.
Le sottoprove saranno distinte dal resto della prova per due caratteristiche:

1. I numeri delle righe delle sottoprove sono preceduti da asterischi, per indicare che tali righe dipendono da un'assunzione.
2. Gli enunciati di una sottoprova hanno il margine sinistro rientrante proporzionalmente al numero di asterischi.

La prova completa di '¬C', a meno di qualche giustificazione, sarà dunque:

1.	¬(G∧C)	: PREMESSA
2.	G	: PREMESSA
*3.	C	: ASSUNZIONE
*4.	G	
*5.	(G∧C)	: ∧INTR,4,3
*6.	¬(G∧C)	
*7.	¬C	: ¬INTR,3,5,6
8.	¬C	

Le righe dalla 3 alla 7 costituiscono la sottoprova di questa derivazione. Una sottoprova si identifica in base al fatto che ogni sua riga ha lo stesso numero di asterischi e che quella parte di derivazione non è interrotta da righe con un numero minore di asterischi. Una sottoprova, inoltre, comincia sempre con la giustificazione

ASSUNZIONE, che in questo esempio compare nella riga 3. La prova principale, costituita da righe prive di asterischi, comprende le righe 1, 2 e 8.

INTRODUZIONE DELLA NEGAZIONE

Nell'argomentazione 6, dall'assunzione di 'C' segue un enunciato che non può in alcun modo essere vero, cioè

$(G \wedge C)$

Esso non può essere vero se le premesse sono tutte vere, perché la sua negazione

$\neg(G \wedge C)$

compare come prima premessa. Quando un enunciato è la negazione di un altro, diciamo che i due enunciati *si contraddicono*: ecco la seconda idea nuova introdotta in questo ragionamento. Secondo la terminologia del Capitolo 7, questi due enunciati non sono simultaneamente soddisfacibili, quindi sono contraddittori. Siccome poi le regole conservano il valore di verità, neppure le premesse e l'assunzione prese insieme sono simultaneamente soddisfacibili, se da esse si derivano enunciati contraddittori. Ciò significa che se le premesse sono soddisfatte, l'assunzione non è soddisfatta, quindi è soddisfatta la negazione dell'assunzione.
Se dunque, in una sottoprova che comincia con un'assunzione, abbiamo derivato due enunciati che si contraddicono, come '$(G \wedge C)$', e la sua negazione '$\neg(G \wedge C)$', possiamo derivare la negazione di tale assunzione, usando $\neg$INTR come giustificazione. Il ragionamento su cui si fonda questa regola è stato spesso usato in filosofia, in matematica, in teologia e in altri campi, quali la giurisprudenza e le scienze naturali. Il nome tradizionale latino è *reductio ad absurdum*, correntemente reso in italiano con "dimostrazione per assurdo". Il concetto di reductio ad absurdum è il seguente: se un ragionamento corretto conduce a due enunciati che non possono essere entrambi veri, allora almeno una delle affermazioni precedenti deve essere sbagliata.
Consideriamo ora in dettaglio l'uso della nuova regola $\neg$INTR della riga 7.

$\neg$INTR Se da un'assunzione **P** si possono derivare, nella stessa sottoprova, sia un enunciato **Q** sia la sua negazione $\neg$**Q**, allora si può derivare

$\neg$**P**

in tale sottoprova.

Si noti che la regola fa riferimento a tre righe precedenti, tutte appartenenti alla stessa sottoprova: viene citata prima la riga 3, che contiene l'assunzione, poi la

riga 5 e infine la riga 6, che contiene la negazione dell'enunciato della riga 5. In questa derivazione mancano le giustificazioni delle righe 4, 6 e 8. L'enunciato della riga 4, nella sottoprova, coincide con l'enunciato della riga 2; inoltre, l'enunciato della riga 6 coincide con quello della riga 1; infine, l'enunciato della riga 8 coincide con quello della riga 7.

LE GIUSTIFICAZIONI 'INVIATO' E 'RESTITUITO'

Una sottoprova deve essere strutturata come una prova (con l'ASSUNZIONE che svolge un ruolo analogo a quello di una PREMESSA), ma deve essere distinta dalla prova principale e da altre sottoprove. Le sottoprove e la prova principale, essendo parti indipendenti della derivazione, devono comunicare fra loro. Il flusso di informazioni fra una sottoprova e la prova principale può avvenire in due direzioni: le informazioni possono essere inviate dalla prova principale alla sottoprova, come avviene, nella derivazione per l'argomentazione 6, fra la riga 2 e la 4 e fra la 1 e la 6, oppure possono essere restituite dalla sottoprova alla prova principale, come avviene fra le righe 7 e 8.
Un flusso lecito di informazioni entrante in una sottoprova verrà giustificato con la nuova regola INVIATO, seguita dalla citazione del numero della riga precedente esterna alla sottoprova. La giustificazione INVIATO può essere usata purché:

1. Ci si trovi in una sottoprova avente più asterischi della riga citata.
2. L'enunciato inviato nella sottoprova sia esattamente quello contenuto nella riga citata.
3. Nessuna delle righe comprese fra la riga giustificata con INVIATO e la riga citata abbia meno asterischi della riga citata.

Un flusso lecito di informazioni uscente da una sottoprova verrà giustificato con la nuova regola RESTITUITO, seguita dalla citazione di una riga della sottoprova. La giustificazione RESTITUITO può essere usata purché:

1. Ci si trovi al di fuori della sottoprova, in una riga avente un asterisco in meno rispetto alla riga citata.
2. L'enunciato restituito sia esattamente quello contenuto nella riga citata.
3. Nessuna riga compresa fra la riga corrente e la riga citata abbia meno asterischi della riga citata.
4. La riga citata abbia la giustificazione $\neg$INTR.

L'ultima condizione assicura che l'informazione restituita non sia contaminata da informazioni derivanti dall'assunzione.
Essendo disponibili queste nuove giustificazioni, la prova di '$\neg$C' diventa ora

una derivazione completa e corretta di '$\neg$C' dalle premesse '$\neg$(G$\wedge$C)' e 'G'.

1.	$\neg$(G$\wedge$C)	: PREMESSA
2.	G	: PREMESSA
*3.	C	: ASSUNZIONE
*4.	G	: INVIATO,2
*5.	(G$\wedge$C)	: $\wedge$INTR,4,3
*6.	$\neg$(G$\wedge$C)	: INVIATO,1
*7.	$\neg$ C	: $\neg$INTR,3,5,6
8.	$\neg$ C	: RESTITUITO,7

Le righe dalla 3 alla 7 costituiscono la sottoprova di questa derivazione. Come tutte le sottoprove, anche questa comincia con un'assunzione. Alle righe 4 e 6 si ha un invio di informazioni alla sottoprova; alla riga 8 si ha una restituzione di informazioni dalla sottoprova alla prova principale.
Regole come INVIATO e RESTITUITO sono poco più che regole di ricopiatura, che permettono di ripetere un enunciato che compare precedentemente in una prova. Queste ricopiature hanno però un significato particolare, in quanto costituiscono una forma di comunicazione fra parti di prova che si trovano a livelli diversi, cioè fra la prova principale e una sottoprova oppure fra una sottoprova e un'altra sottoprova più annidata.
Le regole INVIATO e RESTITUITO, che permettono la comunicazione fra livelli diversi di una derivazione, sono state incluse nel nostro sistema formale di deduzione per diverse ragioni. Innanzitutto, la comunicazione fra una prova principale e una sottoprova assomiglia molto alla comunicazione fra un programma principale e un sottoprogramma in un algoritmo o in un'applicazione informatica. Recenti teorie sullo stile di programmazione hanno sottolineato l'importanza di una dichiarazione esplicita delle informazioni in ingresso e in uscita dai sottoprogrammi.
Le informazioni contenute negli enunciati della prova principale presentano analogie con le informazioni memorizzate nelle cosiddette variabili globali di un programma. Una *variabile globale* è un nome cui è associata un'informazione accessibile o modificabile da qualsiasi punto dell'intero programma, cioè appunto un'informazione globale. Analogamente, le informazioni contenute negli enunciati della prova principale di una derivazione, cioè nelle righe prive di asterischi, possono essere usate ovunque entro una prova, una sottoprova, una sottosottoprova, eccetera, purché vi vengano dapprima inviate. Al contrario, le cosiddette *variabili locali* sono nomi cui sono associate informazioni che possono essere usate solo entro porzioni circoscritte di un algoritmo. Al di fuori di queste porzioni circoscritte, tali variabili non possono assolutamente essere usate; se vengono usate, possono avere valori inammissibili. Le informazioni contenute nelle assunzioni e negli enunciati da esse dipendenti sono analoghe alle informazioni associate alle variabili locali, poiché non possono, in generale, essere usate al di fuori delle sot-

toprove in cui compaiono; le uniche eccezioni sono costituite dalle situazioni in cui si può usare la regola RESTITUITO.
Metaforicamente, possiamo immaginare che le informazioni contenute in una sottoprova siano confinate in essa perché "contaminate" da un enunciato dubbio: l'assunzione. A causa di questo sospetto, non è ammesso che le informazioni di una sottoprova sfuggano nel mondo esterno. Le limitazioni imposte alla comunicazione con il mondo esterno sono disciplinate dalle regole INVIATO e RESTITUITO.

8.9 USO DELLE SOTTOPROVE

Prima di continuare dobbiamo fare diverse osservazioni. Innanzitutto, se si vuole dimostrare che la conclusione di un'argomentazione può essere derivata dalle premesse, è necessario che la conclusione si trovi nella prova principale. Più precisamente, la conclusione deve comparire nell'ultima riga della derivazione, la quale deve essere priva di asterischi. Questa condizione assicura che la conclusione non dipenda da qualche assunzione precedentemente introdotta.
A parte INVIATO e RESTITUITO, inoltre, nessun'altra giustificazione può far riferimento a una riga esterna alla prova o sottoprova in cui compare. In pratica, ciò significa che nessuna riga può mai citare una riga che abbia un numero maggiore o minore di asterischi, a meno che la giustificazione non sia INVIATO o RESTITUITO. Nell'esempio precedente, potrebbe venire spontanea la seguente derivazione:

1. ¬(G∧C)	: PREMESSA
2. G	: PREMESSA
*3. C	: ASSUNZIONE
*4. (G∧C)	: ∧INTR,2,3 [scorretto]
*5. ¬C	: ¬INTR,3,4,1 [scorretto]
6. ¬C	: RESTITUITO,5

Chiariamo i motivi per cui le righe 4 e 5 sono scorrette. La regola ∧INTR non può far riferimento ad una riga esterna alla sua sottoprova, mentre nella riga 4 cita impropriamente la riga 2; l'informazione riguardante 'G' deve invece essere inviata nella sottoprova. Allo stesso modo, anche l'informazione riguardante '¬(G∧C)' deve essere inviata alla sottoprova affinché la regola ¬INTR possa far riferimento ad essa.
Si noti infine che un asterisco viene aggiunto solo con la giustificazione ASSUNZIONE e viene rimosso solo con la giustificazione RESTITUITO.

USO DI ¬INTR

La regola ¬INTR conserva il valore di verità? Supponiamo che le premesse di una derivazione siano tutte vere. Supponiamo inoltre che ¬INTR sia usata per derivare **¬P**, ma che **¬P** abbia, in qualche modo, valore di verità FALSE. Se **¬P** è falso, l'assunzione **P** della sottoprova in cui è usata ¬INTR deve essere vera. Ma se l'assunzione **P** è vera, come possono essere stati ottenuti due enunciati contraddittori? Ci sono solo due modi possibili:

1. La regola INVIATO non conserva il valore di verità.
2. Le altre regole usate, cioè ∧INTR e ∧ELIM, non conservano il valore di verità.

INVIATO, tuttavia, conserva sicuramente il valore di verità, in quanto effettua semplicemente la ricopiatura di un enunciato precedente. Abbiamo inoltre già visto che ∧INTR e ∧ELIM conservano il valore di verità.
Dunque in una sottoprova non si possono ottenere due enunciati contraddittori se tutti gli enunciati precedenti sono veri, a meno che non sia falsa l'assunzione; ma se l'assunzione **P** è falsa deve essere vera la sua negazione **¬P**. Se dunque le premesse sono tutte vere, la sottoprova che usa ¬INTR conduce a un enunciato **¬P** anch'esso vero. Perciò la regola ¬INTR conserva il valore di verità.
Prima di introdurre la prossima regola, consideriamo alcune applicazioni di ¬INTR, di INVIATO e di RESTITUITO.

Argomentazione 7
A
∴ ¬¬A

Derivazione

1. A	: PREMESSA
*2. ¬A	: ASSUNZIONE
*3. A	: INVIATO,1
*4. ¬¬A	: ¬INTR,2,3,2
5. ¬¬A	: RESTITUITO,4

Questa è una prova del fatto che da un enunciato **P** si può derivare la sua doppia negazione **¬¬P**. Si noti che la regola ¬INTR richiede che si aggiunga un segno di negazione all'assunzione; di conseguenza, **P** viene ad acquisire due segni di negazione. Si noti anche che la riga 4 cita due volte la riga 2: la prima volta per citare l'assunzione, la seconda perché la riga 2 è la negazione dell'enunciato della riga 3. Nella derivazione si osservi attentamente l'uso degli asterischi, della marginazione e delle regole INVIATO, RESTITUITO e ¬INTR.

Argomentazione 8
 A
∴ $\neg(\neg A \wedge B)$

Derivazione

1. A	: PREMESSA
*2. $(\neg A \wedge B)$	: ASSUNZIONE
*3. $\neg A$	: $\wedge$ELIM,2
*4. A	: INVIATO,1
*5. $\neg(\neg A \wedge \neg B)$	: $\neg$INTR,2,4,3
6. $\neg(\neg A \wedge \neg B)$	: RESTITUITO,5

8.10 ELIMINAZIONE DELLA NEGAZIONE

La regola *eliminazione della negazione*, o $\neg$ELIM, e la regola $\neg$INTR sono legate da una relazione ancora più stretta di quella che intercorre fra $\wedge$ELIM e $\wedge$INTR; perciò $\neg$ELIM non contiene praticamente nessun concetto che non sia già stato esaminato a proposito di $\neg$INTR.

$\neg$ELIM Se da un'assunzione $\neg$**P** si possono derivare, nella stessa sottoprova, sia un enunciato **Q**, sia la sua negazione $\neg$**Q**, allora si può derivare
 P
in tale sottoprova.

Il parallelismo fra le regole $\neg$INTR e $\neg$ELIM si può evidenziare confrontando i passi da eseguire per applicarle.

$\neg$INTR	**$\neg$ELIM**
In una sottoprova:	In una sottoprova:
Assumere **P**	Assumere $\neg$**P**.
Derivare un enunciato **Q**.	Derivare un enunciato **Q**.
Derivarne la negazione $\neg$**Q**.	Derivarne la negazione $\neg$**Q**.
Derivare con $\neg$INTR la negazione $\neg$**P** dell'assunzione.	Derivare con $\neg$ELIM l'assunzione priva di negazione, cioè **P**.

In altre parole, $\neg$INTR aggiunge (*intr*oduce) un segno di negazione nell'assunzione, mentre $\neg$ELIM rimuove (*elim*ina) un segno di negazione dall'assunzione. I nomi delle regole suggeriscono quando è opportuno applicarle. Se si vuole derivare un enunciato preceduto da un segno di negazione, si può provare ad assumere

l'enunciato privo di negazione, per poi applicare ¬INTR. Se si vuole derivare un enunciato qualsiasi, si può provare ad assumerlo negato, per poi applicare ¬ELIM.
Le norme che regolano le sottoprove e l'uso di INVIATO e RESTITUITO nel caso di ¬INTR valgono anche per ¬ELIM; basta estendere la regola RESTITUITO, ammettendo che essa possa far riferimento non solo ad una riga ottenuta con ¬INTR, ma anche ad una riga ottenuta con ¬ELIM. ¬ELIM presenta dunque ben poche novità.
Consideriamo alcuni esempi.

Argomentazione 9
¬¬A
∴ A

Derivazione

1. ¬¬A	: PREMESSA
*2. ¬A	: ASSUNZIONE
*3. ¬¬A	: INVIATO,1
*4. A	: ¬ELIM,2,2,3
5. A	: RESTITUITO,4

Si consideri attentamente la strategia adottata: per derivare la conclusione 'A' usando ¬ELIM abbiamo assunto '¬A' e abbiamo cercato due enunciati contraddittori.

Argomentazione 10
¬(¬A∧¬B)
¬A
∴ B

Derivazione

1. ¬(¬A∧¬B)	: PREMESSA
2. ¬A	: PREMESSA
*3. ¬B	: ASSUNZIONE
*4. ¬A	: INVIATO,2
*5. (¬A∧¬B)	: ∧INTR,4,3
*6. ¬(¬A∧¬B)	: INVIATO,1
*7. B	: ¬ELIM,3,5,6
8. B	: RESTITUITO,7

Si noti la strategia adottata: per derivare la conclusione 'B' abbiamo assunto la sua negazione '¬B' e abbiamo cercato due enunciati contraddittori.

8.11 RIGHE DI COMMENTO IN UNA DERIVAZIONE

Nelle prove più lunghe, come quella sopra riportata, a volte scriveremo delle note per ricordare che cosa stiamo cercando di fare. Queste note, di grande aiuto nella generazione di una prova, saranno chiamate *commenti* e avranno una struttura del tipo seguente:

/INIZIO: ¬ELIM per derivare B/
/FINE: ¬ELIM per derivare B/

Queste righe non fanno parte della derivazione vera e propria, quindi non sono numerate. I commenti iniziano e finiscono con una barra (/). Se si aggiungono i commenti, la derivazione precedente diventa:

```
1. ¬(¬A∧¬B)    : PREMESSA
2. ¬A          : PREMESSA
   /INIZIO: ¬ELIM per derivare B/
*3. ¬B         : ASSUNZIONE
*4. ¬A         : INVIATO,2
*5. (¬A∧¬B)    : ∧INTR,4,3
*6. ¬(¬A∧¬B)   :INVIATO,1
*7. B          :¬ELIM,3,5,6
   /FINE: ¬ELIM per derivare B/
8. B           : RESTITUITO,7
```

Poiché questi commenti sono molto utili, forniamo qualche indicazione per un loro uso corretto.

1. Ad ogni commento /INIZIO .../ deve corrispondere un commento /FINE .../.
2. È decisamente opportuno usare tali commenti quando si applicano regole che richiedono sottoprove, come ¬INTR e ¬ELIM, in quanto essi evidenziano lo scopo della sottoprova.
3. Quando i commenti sono usati per indicare lo scopo di una sottoprova, devono rientrare dal margine sinistro tanto quanto la sottoprova cui si riferiscono.
4. Quando si genera una derivazione, si deve sempre mirare allo scopo indicato dall'ultimo commento di /INIZIO .../ che è privo del corrispondente commento di /FINE .../.

Questi commenti servono ad indicare lo scopo che ci si prefigge e la regola che si vuole applicare; alcune righe di commento sono perciò utili, mentre troppe finirebbero col celare lo scopo globale della derivazione.
La derivazione seguente illustra un uso complesso di ¬INTR, di ¬ELIM e dei commenti.

Argomentazione 11

$\neg(\neg A \land \neg B)$
$\neg(A \land \neg C)$
$\neg(B \land \neg C)$
$\therefore C$

Derivazione

1. $\neg(\neg A \land \neg B)$: PREMESSA
2. $\neg(A \land \neg C)$: PREMESSA
3. $\neg(B \land \neg C)$: PREMESSA
/INIZIO: $\neg$ELIM per derivare C/
*4. $\neg C$: ASSUNZIONE
/INIZIO: derivare $(\neg A \land \neg B)$ e $\neg(\neg A \land \neg B)$/
/INIZIO: $\neg$INTR per derivare $\neg A$/
**5. A : ASSUNZIONE
**6. $\neg C$: INVIATO,4
**7. $(A \land \neg C)$: $\land$INTR,5,6
**8. $\neg(A \land \neg C)$: INVIATO,2
**9. $\neg A$: $\neg$INTR,5,7,8
/FINE: $\neg$INTR per derivare $\neg A$/
*10. $\neg A$: RESTITUITO,9
/INIZIO: $\neg$INTR per derivare $\neg B$/
**11. B : ASSUNZIONE
**12. $\neg C$: INVIATO,4
**13. $(B \land \neg C)$: $\land$INTR,11,12
**14. $\neg(B \land \neg C)$: INVIATO,3
**15. $\neg B$: $\neg$INTR,11,13,14
/FINE: $\neg$INTR per derivare $\neg B$/
*16. $\neg B$: RESTITUITO,15
*17. $(\neg A \land \neg B)$: $\land$INTR,10,16
*18. $\neg(\neg A \land \neg B)$: INVIATO,1
/FINE: derivare $(\neg A \land \neg B)$ e $\neg(\neg A \land \neg B)$/
*19. C : $\neg$ELIM,4,17,18
/FINE: $\neg$ELIM per derivare C/
20. C : RESTITUITO,19

Il fondamento della strategia qui adottata consiste, come il primo commento illustra, nell'applicare $\neg$ELIM per tentare di derivare la conclusione ‘C’. Una volta assunto ‘$\neg$C’, però, occorre cercare due enunciati contraddittori, che potrebbero essere

$(\neg A \land \neg B)$

e

$\neg(\neg A \wedge \neg B)$

Il secondo enunciato è semplicemente la prima premessa. Ma per derivare '$(\neg A \wedge \neg B)$' occorre derivare prima '$\neg A$' e poi '$\neg B$', per unirli quindi con ∧INTR. La strategia adottata può essere delineata riportando i soli commenti.

/INIZIO: ¬ELIM per derivare C/
/INIZIO: derivare $(\neg A \wedge \neg B)$ e $\neg(\neg A \wedge \neg B)$/
    /INIZIO: ∧INTR per derivare $(\neg A \wedge \neg B)$/
        /INIZIO: ¬INTR per derivare ¬A/
        /FINE: ¬INTR per derivare ¬A/
        /INIZIO: ¬INTR per derivare ¬B/
        /FINE: ¬INTR per derivare ¬B/
    /FINE: ∧INTR per derivare $(\neg A \wedge \neg B)$/
/FINE: derivare $(\neg A \wedge \neg B)$ e $\neg(\neg A \wedge \neg B)$/
/FINE: ¬ELIM per derivare C/

8.12 COMPLETEZZA

Sotto un certo punto di vista, il sistema formale di deduzione fin qui esposto, comprendente solo le regole, o giustificazioni

PREMESSA	∧INTR
ASSUNZIONE	∧ELIM
INVIATO	¬INTR
RESTITUITO	¬ELIM

si può considerare completo. Si noti innanzitutto che qualunque enunciato contenente i connettivi proposizionali ∧, ¬, ∨, →, NOR, NAND, eccetera, è logicamente equivalente ad un enunciato contenente solo i connettivi ¬ e ∧. Ad esempio:

$(\mathbf{P} \vee \mathbf{Q})$ è logicamente equivalente a $\neg(\neg \mathbf{P} \wedge \neg \mathbf{Q})$
$(\mathbf{P} \rightarrow \mathbf{Q})$ è logicamente equivalente a $\neg(\mathbf{P} \wedge \neg \mathbf{Q})$.

È quindi possibile convertire ogni enunciato che usi questi connettivi in un enunciato logicamente equivalente che contenga solo i connettivi ¬ e ∧. Esiste anche un algoritmo per effettuare questa conversione, che però non sarà descritto qui (vedi Esercizio F).

Inoltre, le coppie di regole

$\wedge$INTR
$\wedge$ELIM

e

$\neg$INTR
$\neg$ELIM

sono sufficienti per derivare qualunque conclusione che segua validamente da un insieme di premesse, purché le premesse e la conclusione vengano espresse usando solo i connettivi $\neg$ e $\wedge$; di questo fatto non forniremo però la dimostrazione in questa sede.
Poiché però delle derivazioni che usassero solo queste regole risulterebbero spesso molto lunghe e difficili da costruire, soprattutto a causa delle complesse regole $\neg$INTR e $\neg$ELIM, nel prossimo capitolo introdurremo delle regole di interferenza aggiuntive, direttamente applicabili ad enunciati contenenti $\vee$, $\rightarrow$ e $\leftrightarrow$.

8.13 CONCLUSIONI

Dopo aver introdotto il concetto di *sistema formale*, abbiamo definito *sistema di deduzione naturale* un sistema formale i cui elementi sono enunciati e le cui regole conservano il valore di verità. Il sistema di deduzione naturale finora sviluppato comprende le regole seguenti, che descriviamo qui in modo breve e informale:

PREMESSA	Le premesse sono poste all'inizio di una derivazione.
ASSUNZIONE	Qualunque enunciato può essere assunto come prima riga di una sottoprova.
$\wedge$INTR	Se **P** e **Q** sono righe precedenti. allora si può derivare (**P**$\wedge$**Q**).
$\wedge$ELIM	Se (**P**$\wedge$**Q**) è una riga precedente allora si può derivare **P** e **Q**.
$\neg$INTR	Se l'assunzione **P** porta a una contraddizione allora si può derivare $\neg$**P**.
$\neg$ELIM	Se l'assunzione $\neg$**P** porta a una contraddizione allora si può derivare **P**.
INVIATO	Qualunque riga precedente di una prova può essere inviata a una sottoprova.
RESTITUITO	Sotto opportune condizioni, l'ultima riga di una sottoprova può essere portata fuori dalla sottoprova stessa.

Nel nostro sistema, una derivazione è una sequenza numerata di righe contenenti ciascuna un enunciato e la sua giustificazione. Abbiamo operato una distinzione fra *prova principale* e *sottoprove*. Infine, abbiamo introdotto l'uso dei *commenti*, che aiutano a strutturare la derivazione, ma formalmente non ne fanno parte.

8.14 ESERCIZI

A. Gioco delle stelle e delle barre
  1. Perché '/B☆A' è una stringa perdente?
  2. Perché 'A☆B' è una stringa perdente?
  3. La stringa S del paragrafo 8.1 è vincente? Se lo è, come la si può costruire? Se è perdente, perché lo è?

B. Sistemi formali
  1. Si voglia definire un nuovo gioco, "Sinistra e destra", nel quale, per le stringhe vincenti, si distinguono le barre di sinistra da quelle di destra, rappresentando le prime con '(' e le seconde con ')'.
     a. Come deve essere modificata la regola 3?
     b. Costruire due stringhe vincenti e due perdenti nel gioco "Sinistra e destra". Spiegare come vengono derivate le stringhe vincenti e perché le stringhe perdenti non possono essere derivate.
  2. a. Scrivere un insieme completo di regole per un'ulteriore modifica, nella quale il simbolo ☆ è sostituito dai simboli di somma, sottrazione, moltiplicazione e divisione. Chiamiamo questo nuovo sistema "Gioco delle formule aritmetiche".
     b. Costruire due stringhe vincenti e due perdenti nel gioco delle formule aritmetiche. Spiegare come vengono derivate le stringhe vincenti e perché le stringhe perdenti non possono essere derivate.
  3. Modificare ulteriormente il sistema, aggiungendo un nuovo simbolo che rappresenti un operatore prefisso, come il simbolo $\sqrt{\ }$ di radice quadrata, o il simbolo $-$, usato per formare l'opposto di un numero.

C. Sistema di deduzione naturale
   Usando il sistema di deduzione naturale definito in questo capitolo, derivare le conclusioni delle argomentazioni seguenti:

1. $(A \wedge B)$
   $\neg(A \wedge \neg C)$
   $\therefore C$
2. A
   $\therefore \neg(\neg B \wedge B)$
3. $\neg(A \wedge \neg B)$
   $\neg B$
   $\therefore \neg A$
4. $(A \wedge \neg A)$
   $\therefore B$
5. $\neg(A \wedge \neg B)$
   A
   $\therefore B$

D. Rappresentare simbolicamente le argomentazioni seguenti e fornirne le derivazioni:
  1. Non si dà il caso che, allo stesso tempo, Alfredo sia un artista e Carlo sia uno scienziato. Poiché Alfredo è un artista, Carlo non è uno scienziato.

2. Non si dà il caso che, allo stesso tempo, Davide abbia fame e Patrizia no. Ma Patrizia non ha fame. Perciò Davide non ha fame.

E. La conclusione di una derivazione priva di premesse è un teorema. Per dimostrare che un enunciato è un teorema, si comincia con un'assunzione, cosicché l'unica riga della prova principale è l'ultima. Fornire le derivazioni dei seguenti teoremi.
[*Suggerimento*: usare ¬INTR per ognuno di essi.]
   1. $\neg(A \wedge \neg A)$
   2. $\neg((A \wedge B) \wedge \neg(A \wedge B))$
   3. $\neg((\neg(A \wedge \neg B) \wedge A) \wedge \neg B)$

F. Equivalenze logiche
   1. Definire un algoritmo per sostituire ciascuno degli enunciati seguenti con un enunciato logicamente equivalente, contenente solo i connettivi ¬ e ∧.

      $(\mathbf{P} \rightarrow \mathbf{Q})$
      $(\mathbf{P} \vee \mathbf{Q})$
      $(\mathbf{P} \leftrightarrow \mathbf{Q})$
      (**P** NOR **Q**)
      (**P** NAND **Q**)

   2. Considerare un enunciato arbitrario, contenente solo i connettivi ¬, ∧, ∨, → e ↔. Definire un algoritmo che determini un enunciato logicamente equivalente, contenente solo i connettivi ¬ e ∧.
   3. a. Esprimere $(\mathbf{P} \wedge \mathbf{Q})$ usando solo NOR.
      b. Esprimere $\neg\mathbf{P}$ usando solo NOR.
      c. Esprimere $(\mathbf{P} \wedge \mathbf{Q})$ usando solo NAND.
      d. Esprimere $\neg\mathbf{P}$ usando solo NAND.

Al punto 1 di questo esercizio abbiamo visto che tutti gli enunciati possono essere espressi in termini di ¬ e di ∧ mentre al punto 3 abbiamo visto che tutti gli enunciati contenenti solo ¬ e ∧ possono essere espressi usando solo NOR o solo NAND; se ne può concludere che *tutti* gli enunciati possono essere espressi usando *un* solo connettivo binario, che può essere NOR oppure NAND.

9

Logica enunciativa: regole di inferenza supplementari

Nel capitolo precedente abbiamo studiato due coppie di regole per derivare enunciati da altri enunciati. Come si è detto, queste quattro regole, cioè ∧INTR, ∧ELIM, ¬INTR e ¬ELIM, permettono di derivare tutti gli enunciati deducibili con validità da premesse assegnate. Esistono però due fattori che rendono poco opportuno l'uso di queste sole quattro regole. Innanzitutto, le regole ¬INTR e ¬ELIM richiedono l'individuazione di due enunciati contraddittori, ma non esiste un criterio semplice per definire in anticipo quali essi debbano essere. Per applicare ¬INTR e ¬ELIM occorre perciò avanzare delle ipotesi sui possibili enunciati contraddittori, o addirittura affrontare una sottoprova alla cieca, nella speranza di scoprirne per caso due.

Il secondo fattore è un ostacolo ancora maggiore, se si usano solo le quattro regole fin qui studiate, poiché esse permettono di derivare altri enunciati solo se vengono applicate ad enunciati contenenti unicamente congiunzioni e negazioni. Tuttavia, come abbiamo visto nei Capitoli 3 e 4, è utile usare altri connettivi, come ∨, →, ↔ e altri ancora. Si potrebbe tradurre ogni enunciato contenente questi connettivi supplementari in un enunciato logicamente equivalente, che contenga solo i connettivi di negazione e di congiunzione; è però più semplice introdurre regole apposite per gli altri connettivi. In questo capitolo amplieremo il nostro sistema di deduzione, aggiungendovi delle regole supplementari; queste regole non sono strettamente necessarie, ma spesso permettono di costruire una derivazione molto più semplicemente di quanto non sia possibile usando solo le quattro regole del Capitolo 8.

9.1 INTRODUZIONE ED ELIMINAZIONE DEL CONDIZIONALE

Uno dei tipi di ragionamento più usati è:

> Se **P**, allora **Q**
> **P**.
> Perciò **Q**.

Ad esempio:

> Se il maggiordomo aveva un movente, allora è il maggiordomo che ha commesso il delitto.
> Il maggiordomo aveva un movente.
> Perciò è il maggiordomo che ha commesso il delitto.

È noto fin dall'antichità che questo tipo di ragionamento è corretto. In latino esso è detto *modus ponens*. In forma simbolica, il modello di questa argomentazione si può rappresentare così:

$$\begin{array}{l} (\mathbf{P} \rightarrow \mathbf{Q}) \\ \mathbf{P} \\ \therefore \mathbf{Q} \end{array}$$

Poiché l'enunciato $(\mathbf{P} \rightarrow \mathbf{Q})$ è logicamente equivalente a $\neg(\mathbf{P} \wedge \neg \mathbf{Q})$, l'argomentazione sopra riportata è valida se e solo se l'argomentazione

$$\begin{array}{l} \neg(\mathbf{P} \wedge \neg \mathbf{Q}) \\ \mathbf{P} \\ \therefore \mathbf{Q} \end{array}$$

è valida. Nel capitolo precedente avevamo fornito la seguente derivazione di un'argomentazione avente proprio questa forma:

1.	$\neg(\mathbf{P} \wedge \neg \mathbf{Q})$	: PREMESSA
2.	$\mathbf{P}$	: PREMESSA
	/INIZIO: $\neg$ELIM per derivare **Q**/	
*3.	$\neg \mathbf{Q}$	: ASSUNZIONE
*4.	$\mathbf{P}$	: INVIATO,2
*5.	$(\mathbf{P} \wedge \neg \mathbf{Q})$	: $\wedge$INTR,4,3
*6.	$\neg(\mathbf{P} \wedge \neg \mathbf{Q})$	: INVIATO,1
*7.	$\mathbf{Q}$	: $\neg$ELIM,3,5,6
	/FINE: $\neg$ELIM per derivare **Q**/	
8.	$\mathbf{Q}$	: RESTITUITO,7

Poiché **(P→Q)** è logicamente equivalente a **¬(P∧¬Q)**, e poiché il modello di argomentazione **¬(P∧¬Q), P∴Q** conserva la verità, anche il modello **(P→Q), P∴Q** deve conservare la verità. Su questa considerazione si fonda la nuova regola *eliminazione del condizionale*, o →ELIM.

→ELIM Da un enunciato del tipo

  (P→Q)

 e da un enunciato del tipo

  P

 si può derivare l'enunciato

  Q

Per verificare in modo più diretto che questa regola conserva la verità, si può esaminare la tavola di verità di **(P→Q)**: non esiste nessuna situazione in cui **P** e **(P→Q)** siano entrambi veri e **Q** sia falso.
Consideriamo ora diverse applicazioni della regola →ELIM nelle derivazioni.

Argomentazione 1

(A→(B→C))
A
∴(B→C)

Derivazione

1. (A→(A→C)) : PREMESSA
2. A : PREMESSA
3. (B→C) : →ELIM,1,2

In questo caso la conclusione, '(B→C)', è a sua volta un condizionale.
Prima di applicare la regola →ELIM occorre accertarsi che i due enunciati abbiano la forma richiesta, cioè:

(<enunciato 1>→<enunciato 2>)
<enunciato 2>
Perciò <enunciato 1>

Nel Capitolo 6 abbiamo però visto che questo modello di argomentazione non è valido; perciò una regola basata su di esso non conserverebbe la verità. Questo tipo di ragionamento che non conserva la verità viene spesso erroneamente usato, ed è detto *fallacia*; la particolare fallacia qui riportata è l'*affermazione del conseguente*.
L'altra regola, oltre a →ELIM, che si riferisce al condizionale è l'*introduzione del condizionale*, o →INTR.

→INTR Se da un'assunzione

  P

si può derivare un enunciato
Q
nella stessa sottoprova, allora si può derivare
(P→Q)

La regola →INTR, come già le regole ¬INTR e ¬ELIM, richiede una sottoprova che cominci con un'assunzione. Il ragionamento intuitivo su cui si fonda questa regola non è difficile da capire. Supponiamo che un enunciato **P** sia vero. Se l'enunciato **P** si può derivare assumendo **P**, abbiamo dimostrato **Q** sotto la condizione **P**, cioè: se **P**, allora **Q**.
La regola →ELIM si usa per ridurre il numero di connettivi condizionali, mentre →INTR si usa per derivare un enunciato il cui connettivo principale sia →. Consideriamo alcune applicazioni di quest'ultima regola:

Argomentazione 2

(A→B)
(B→C)
∴(A→C)

Derivazione

1.	(A→B)	: PREMESSA
2.	(B→C)	: PREMESSA
*3.	A	: ASSUNZIONE
*4.	(A→B)	: INVIATO,1
*5.	(B→C)	: INVIATO,2
*6.	B	: →ELIM,4,3
*7.	C	: →ELIM,5,6
*8.	(A→C)	: →INTR,3,7
9.	(A→C)	: RESTITUITO,8

Prima di cominciare la derivazione notiamo che la conclusione '(A→C)' è un condizionale. Di solito, quando si vuole derivare un condizionale, si procede in modo da poter applicare →INTR. Per applicare →INTR dobbiamo innanzitutto introdurre un'assunzione, che dev'essere l'antecedente del condizionale che intendiamo derivare. Una volta iniziata la sottoprova con l'assunzione, dobbiamo derivare il conseguente del condizionale desiderato. Quando abbiamo a disposizione sia l'assunzione, sia il conseguente del condizionale, possiamo infine applicare →INTR, ottenendo così il risultato voluto. Si noti che →INTR fa riferimento a due righe precedenti, che devono appartenere entrambe alla stessa sottoprova: la prima riga citata contiene l'assunzione, mentre la seconda contiene il conseguente del condizionale desiderato. Si osservi anche che la regola RESTITUITO è stata estesa, ammettendo che essa possa far riferimento ad una riga in cui è stata applicata la regola →INTR.

La strategia adottata in questa derivazione può essere evidenziata aggiungendo i commenti:

1. $(A \to B)$: PREMESSA
2. $(B \to C)$: PREMESSA
/INIZIO: $\to$INTR per derivare $(A \to C)$/
*3. A : ASSUNZIONE
/INIZIO: $\to$ELIM per derivare C/
*4. $(A \to B)$: INVIATO,1
*5. $(B \to C)$: INVIATO,2
*6. B : $\to$ELIM,4,3
*7. C : ELIM,5,6
/FINE: $\to$ELIM per derivare C/
*8. $(A \to C)$: INTR,3,7
/FINE: $\to$INTR per derivare $(A \to C)$/
9. $(A \to C)$: RESTITUITO,8

Consideriamo un altro esempio.

Argomentazione 3

$(A \wedge B)$
$\therefore (C \to A)$

È evidente che in questo caso si deve usare $\to$INTR, perché la conclusione contiene un connettivo $\to$, che non compare invece nella premessa. Per ottenere il risultato corretto applicando $\to$INTR bisogna assumere 'C' e ricavare 'A' nella corrispondente sottoprova.

Derivazione

1. $(A \wedge B)$: PREMESSA
/INIZIO: $\to$INTR per derivare $(C \to A)$/
*2. C : ASSUNZIONE
*3. $(A \wedge B)$: INVIATO,1
*4. A : $\wedge$ELIM,3
*5. $(C \to A)$: $\to$INTR,2,4
/FINE: $\to$INTR per derivare $(C \to A)$/
6. $(C \to A)$: RESTITUITO,5

Osserviamo che la conclusione è un condizionale, da ottenere quindi, con ogni probabilità, mediante $\to$INTR. Se vogliamo applicare $\to$INTR per ottenere '$(C \to A)$', dobbiamo assumere 'C' per poi, in qualche modo, derivare 'A'. Nella riga 2 abbiamo assunto 'C'. In pratica, a questo punto, il solo passaggio possibile è l'invio dell'unica premessa nella sottoprova. Una volta fatto ciò, è evidentemente facile derivare 'A'.

Argomentazione 4

A
∴ (B→(C→A))

Derivazione

1. A : PREMESSA
/INIZIO: →INTR per derivare (B→(C→A))/
*2. B : ASSUNZIONE
/INIZIO: →INTR per derivare (C→A)/
**3. C : ASSUNZIONE
**4. A : INVIATO,1
**5. (C→A) : →INTR,3,4
/FINE: →INTR per derivare (C→A)/
*6. (C→A) : RESTITUITO,5
*7. (B→(C→A)) : INTR,2,6
/FINE: →INTR per derivare (B→(C→A))/
8. (B→(C→A)) : RESTITUITO,7

Questa derivazione usa due applicazioni di →INTR; di conseguenza c'è una sottoprova entro una sottoprova. La conclusione desiderata è '(B→(C→A))'; poiché si tratta di un condizionale, la regola adatta appare essere →INTR. Per giungere a questa conclusione dobbiamo assumere

B

che è l'antecedente della conclusione, e cercare di ottenere

(C→A)

che ne è il conseguente. Ma '(C→A)' è a sua volta un condizionale; ciò suggerisce di applicare nuovamente →INTR. Per derivare '(C→A)' applichiamo dunque →INTR, cominciando con l'assunzione dell'enunciato 'C' e cercando di ottenere l'enunciato 'A'. La strategia complessiva è:

/INIZIO: →INTR per derivare (B→(C→A))/
/INIZIO: →INTR per derivare (C→A)/
/FINE: →INTR per derivare (C→A)/
/FINE: →INTR per derivare (B→(C→A))/

Questa strategia praticamente impone la forma finale della derivazione. Ad esempio, il commento:

/INIZIO: →INTR per derivare (B→(C→A))/

impone che il prossimo passaggio sia l'assunzione dell'antecedente 'B'.

9.2 INTRODUZIONE ED ELIMINAZIONE DELLA DISGIUNZIONE

Esiste una coppia di regole riguardanti le disgiunzioni; si tratta dell'*introduzione della disgiunzione*, o $\vee$INTR, e dell'*eliminazione della disgiunzione*, o $\vee$ELIM. La regola $\vee$INTR è estremamente semplice.

$\vee$INTR Da un enunciato qualunque
  $\mathbf{P}$
si può derivare
  $(\mathbf{P} \vee \mathbf{Q})$
oppure, a scelta,
  $(\mathbf{Q} \vee \mathbf{P})$

Essa afferma che da un qualunque enunciato **P** si può derivare un enunciato costruito "aggiungendo" a **P** un altro enunciato **Q**.
L'enunciato $\rightarrow$, che viene aggiunto all'enunciato **P** già disponibile, può essere *un enunciato qualsiasi*; perciò le righe dalla 2 alla 5 sono tutte applicazioni lecite di $\vee$INTR:

1. A : PREMESSA
2. $(A \vee B)$: $\vee$INTR,1
3. $(A \vee (C \wedge D))$: $\vee$INTR,1
4. $((E \rightarrow (F \vee G)) \vee A)$: $\vee$INTR,1
5. $((A \vee B) \vee (A \vee B))$: $\vee$INTR,2

Nella prima riga l'enunciato aggiunto è 'B'. Nella terza riga è l'enunciato molecolare '$(C \wedge D)$'. Nella quarta riga, all'enunciato 'A' viene aggiunto l'enunciato molecolare

$$(E \rightarrow (F \vee G))$$

ma 'A' compare come secondo disgiunto. Nella quinta riga, '$(A \vee B)$' viene aggiunto a se stesso.
La regola $\vee$INTR ha una produttività esuberante e può essere applicata a qualsiasi enunciato; la difficoltà non consiste perciò nel capire la regola, bensì nell'imparare quando e come usarla e quali enunciati aggiungere. Come nel caso delle altre regole, il nome stesso della regola $\vee$INTR fornisce qualche indicazione sui casi in cui può convenire usarla: la regola $\vee$INTR di solito va applicata quando si deve derivare una disgiunzione. Si consideri ad esempio la seguente argomentazione valida:

Argomentazione 5

  $(A \wedge B)$
$\therefore (C \rightarrow (A \vee D))$

La conclusione desiderata è un condizionale: (C→(A∨D)). La strategia dovrà quindi fondarsi sull'applicazione di →INTR. Cominciamo perciò con l'assumere 'C'. Dopo ciò dobbiamo però derivare '(A∨D)', cioè una disgiunzione; appare quindi logico usare ∨INTR. Adotteremo dunque la strategia seguente:

/INIZIO: →INTR per derivare (C→(A∨D))/
/INIZIO: ∨INTR per derivare (A∨D)/
/FINE: ∨INTR per derivare (A∨D)/
/FINE: →INTR per derivare (C→(A∨D))/

La derivazione che ne risulta è:

1. (A∧B) : PREMESSA
/INIZIO: →INTR per derivare (C→(A∨D))/
*2. C : ASSUNZIONE
/INIZIO: ∨INTR per derivare (A∨D)/
*3. (A∧B) : INVIATO,1
*4. A : ∧ELIM,3
*5. (A∨D) : ∨INTR,4
/FINE: ∨INTR per derivare (A∨D)/
*6. (C→(A∨D)) : →INTR,2,5
/FINE:→INTR per derivare (C→(A∨D))/
7. (C→(A∨D)) : RESTITUITO,6

Al passo 5 avremmo potuto aggiungere ad 'A' un enunciato del tutto arbitrario; siccome però intendevamo derivare '(A∨D)', era logico aggiungere 'D'.
Vediamo un'altra applicazione di ∨INTR:

Argomentazione 6

((A∨B)→C)
A
∴C

Derivazione

1. ((A∨B)→C) : PREMESSA
2. A : PREMESSA
3. (A∨B) : ∨INTR,2
4. C : →ELIM,1,3

Anche la regola ∨ELIM è semplice da definire:

∨ELIM Da un enunciato del tipo
(P∨Q)

e da uno del tipo
$\neg \mathbf{P}$
si può derivare
$\mathbf{Q}$

o, equivalentemente, da un enunciato del tipo
$(\mathbf{P} \vee \mathbf{Q})$
e da uno del tipo
$\neg \mathbf{Q}$
si può derivare
$\mathbf{P}$

In altre parole, da una disgiunzione e dalla negazione di uno dei disgiunti si può derivare l'altro disgiunto.
L'esempio più semplice di applicazione di questa regola è il seguente:

Argomentazione 7

$(A \vee B)$
$\neg A$
$\therefore B$

Derivazione

1. $(A \vee B)$: PREMESSA
2. $\neg A$: PREMESSA
3. B : $\vee$ELIM,1,2

Si noti che la regola $\vee$ELIM cita prima la riga contenente la disgiunzione, poi la riga contenente il disgiunto negato.
Vediamo un'altra applicazione di $\vee$ELIM:

Argomentazione 8

$(B \vee C)$
$(A \rightarrow \neg C)$
$\therefore (A \rightarrow B)$

A prima vista, questa argomentazione può non sembrare molto adatta a un'applicazione di $\vee$ELIM. Le premesse sono una disgiunzione, '$(B \vee C)$', e un condizionale, '$(A \rightarrow \neg C)$'. La conclusione voluta è un condizionale, quindi si può tentare una strategia fondata su $\rightarrow$INTR, assumendo 'A' e cercando poi di derivare 'B' nella sottoprova.

Derivazione

1. $(B \vee C)$: PREMESSA

2. (A→¬C) : PREMESSA
/INIZIO: →INTR per derivare (A→B)/
*3. A : ASSUNZIONE
*4. (A→¬C) : INVIATO,2
*5. ¬C : →ELIM,4,3
*6. (B∨C) : INVIATO,1
*7. B : ∨ELIM,6,5
*8. (A→B) : →INTR,3,7
/FINE: →INTR per derivare (A→B)/
9. (A→B) : RESTITUITO,8

Si noti che nell'esempio seguente ∨ELIM è usato in modo non corretto:

Argomentazione 9

(B∨¬C)
C
∴B

Derivazione

1. (B∨¬C) : PREMESSA
2. C : PREMESSA
3. B : ∨ELIM,1,2 [scorretto]

Nel seguito di questo capitolo vedremo come si deriva '¬ ¬C' da 'C'; la derivazione corretta della conclusione è allora:

1. (B∨¬C) : PREMESSA
2. C : PREMESSA
3. ¬ ¬C : RS DN,2 (vedi "Regole di sostituzione")
4. B : ∨ELIM,1,3

9.3 CONSERVAZIONE DELLA VERITÀ

Le ultime quattro regole di inferenza introdotte (→ELIM, →INTR, ∨INTR e ∨ELIM) conservano la verità? Esistono due metodi per rispondere a questa domanda. Si potrebbe, ad esempio, tentare di dimostrare direttamente che una regola nuova non produrrà mai enunciati falsi a partire da enunciati veri; a tal fine, si potrebbe esaminare una tavola di verità per vedere se esiste una situazione in cui gli enunciati precedenti siano veri ma quello derivato sia falso. Un altro metodo, di solito più semplice, consiste nel dimostrare che la regola nuova è derivabile da regole già note, che si sa che conservano la verità. Diremo che una regola nuova è derivabile da regole vecchie se la regola nuova permette di derivare solo enun-

ciati derivabili anche con le regole vecchie.
Si vede subito che la regola ∨INTR è derivabile da una combinazione di altre regole già introdotte, che, come abbiamo dimostrato, conservano tutte la verità. Lo schema di inferenza di ∨INTR è:

$$\mathbf{P}$$
$$\therefore (\mathbf{P} \lor \mathbf{Q})$$

dove **Q** può essere un enunciato qualsiasi. Applichiamo dunque il secondo metodo per verificare se questa regola conserva la verità: ci chiediamo se è possibile passare da **P** a (**P**∨**Q**) usando solo le regole introdotte nel capitolo precedente. La risposta è affermativa, come mostra la seguente derivazione (ripresa appunto dal capitolo precedente):

1.	**P**	: PREMESSA	
*2.	(¬**P**∧¬**Q**)	: ASSUNZIONE	[¬(¬**P**∧¬**Q**) è logicamente equivalente a (**P**∨**Q**)]
*3.	**P**	: INVIATO,1	
*4.	¬**P**	: ∧ELIM,2	
*5.	¬(¬**P**∧¬**Q**)	: ¬INTR,2,3,4	
6.	¬(¬**P**∧¬**Q**)	: RESTITUITO,5	

In altre parole, ogni volta che, in una derivazione, si ottiene un enunciato **P**, si può inserire la derivazione sopra riportata (modificandone opportunamente i numeri di riga), per derivare ¬(¬**P**∧¬**Q**) senza usare ∨INTR.
In modo analogo si può dimostrare che ∨ELIM conserva la verità. Dimostrare invece che →INTR conserva la verità è difficile con il secondo metodo, ma è piuttosto facile con il primo.

9.4 INTRODUZIONE ED ELIMINAZIONE DEL BICONDIZIONALE

↔ è l'ultimo connettivo di uso frequente per il quale dobbiamo ancora definire regole di inferenza che conservino la verità. Una prima regola relativa al bicondizionale è:

↔INTR Da un enunciato del tipo
(**P**→**Q**)
e da uno del tipo
(**Q**→**P**)
si può derivare
(**P**↔**Q**)

Si noti che l'applicazione della regola ↔INTR richiede due enunciati precedenti, entrambi condizionali, tali che l'antecedente dell'uno sia il conseguente dell'altro. L'altra regola relativa al bicondizionale:

↔ELIM Da un enunciato del tipo
(P↔Q)
si può derivare
(P→Q)
oppure, a scelta,
(Q→P)

Queste due regole, come i nomi stessi suggeriscono, si applicano di solito quando si deve derivare un enunciato contenente ↔, oppure quando da un enunciato contenente il connettivo ↔ si deve derivare un enunciato che ne sia privo. L'applicazione di queste due regole è illustrata nelle due derivazioni seguenti.

Argomentazione 10

(A↔B)
A
∴B

Derivazione

1. (A↔B) : PREMESSA
2. A : PREMESSA
/INIZIO: ↔ELIM per ottenere (A→B)/
3. (A→B) : ↔ELIM,1
/FINE: ↔ELIM per ottenere (A→B)/
4. B : →ELIM,3,2

Si noti che l'uso di ↔ELIM in una derivazione richiede la citazione di una sola riga precedente, la quale deve contenere un bicondizionale. Nell'argomentazione sopra riportata, è evidente che si deve applicare ↔ELIM, perché una delle premesse contiene un bicondizionale, mentre la conclusione ne è priva; si deve quindi eliminare il connettivo ↔.

Argomentazione 11

(A∧(B→C))
(C→B)
∴(A∧(B↔C))

Derivazione

1. (A∧(B→C)) : PREMESSA
2. (C→B) : PREMESSA

3. A : $\wedge$ELIM,1
4. $(B \to C)$: $\wedge$ELIM,1
5. $(B \leftrightarrow C)$: $\leftrightarrow$INTR,4,2
6. $(A \wedge (B \leftrightarrow C))$: $\wedge$INTR,3,5

È facile dimostrare che queste due regole conservano la verità.

9.5 REGOLE DI SOSTITUZIONE

Tutte le regole finora viste sono, per così dire, a senso unico; esse permettono infatti di derivare un enunciato a partire da enunciati precedenti:

<enunciato 1>
<enunciato 2>
.
.
.

deriviamo:
<enunciato 3>

Ciò non significa però che da <enunciato 3> si possa derivare <enunciato 1>. Ad esempio, da $(\mathbf{P} \wedge \mathbf{Q})$ si può derivare $\mathbf{P}$, ma dal solo enunciato $\mathbf{P}$ non si può derivare $(\mathbf{P} \wedge \mathbf{Q})$. Inoltre, queste regole si possono applicare solo quando *un intero enunciato* di una riga ha la forma appropriata. Ad esempio, da

1. $(A \to B)$: PREMESSA
2. A : PREMESSA

possiamo derivare, con $\to$ELIM,

3. B : $\to$ELIM,1,2

giacché gli enunciati delle righe 1 e 2 hanno la forma corretta. Si noti invece che la derivazione seguente non è corretta:

1. $((A \to B) \vee C)$: PREMESSA
2. A : PREMESSA
3. B : $\to$ELIM,1,2 [scorretto]

Il problema è che, benché la riga 1 contenga il condizionale '$(A \to B)$', l'intero enunciato della riga 1 non è un condizionale, bensì una disgiunzione. Insomma,

tutte le regole di inferenza finora introdotte si possono applicare ad enunciati precedenti della derivazione, ma non a parti di tali enunciati.
Ad esempio, si potrebbe essere tentati di effettuare la derivazione seguente:

1. ((A∧B)→C) : PREMESSA
2. (A→C) : ∧ELIM,1 [scorretto]

In questo caso, il problema consiste nel fatto che ∧ELIM si può applicare solo ad un enunciato che *sia* una congiunzione, ma non a un enunciato che *contenga* una congiunzione.
In quest'ultimo paragrafo introdurremo delle regole che sono "bidirezionali" e che permettono di alterare parti di enunciati precedenti: si tratta delle *regole di sostituzione* (RS).

RS Da ogni enunciato, una cui parte sia un enunciato **P**:
(... **P** ...)
se **P** è equivalente a **Q**, si può derivare lo stesso enunciato, mettendo **Q** al posto di **P**:
(... **Q** ...)

In altre parole, si può derivare un enunciato sostituendo un enunciato componente con un enunciato logicamente equivalente. In teoria, prima di applicare la regola di sostituzione, sostituendo un enunciato componente con uno ad esso equivalente, bisognerebbe dimostrare l'equivalenza logica di tali due enunciati. Esistono però alcune coppie di enunciati equivalenti di uso molto frequente; possiamo allora attribuire un nome a ciascuna di esse, per poi farvi riferimento quando occorre, senza doverne dimostrare ogni volta l'equivalenza logica.
Queste coppie di enunciati equivalenti sono:

P	¬¬**P**	DN (doppia negazione)
(**P**∧**Q**)	(**Q**∧**P**)	CM (commutatività di ∧)
(**P**∨**Q**)	(**Q**∨**P**)	CM (commutatività di ∨)
P	(**P**∧**P**)	ID (idempotenza di ∧)
P	(**P**∨**P**)	ID (idempotenza di ∨)
(**P**∧(**Q**∧**R**))	((**P**∧**Q**)∧**R**)	AS (associatività di ∧)
(**P**∨(**Q**∨**R**))	((**P**∨**Q**)∨**R**)	AS (associatività di ∨)
(**P**∧(**Q**∨**R**))	((**P**∧**Q**)∨(**P**∧**R**))	DIS (distributività di ∧ rispetto a ∨)
(**P**∨(**Q**∧**R**))	((**P**∨**Q**)∧(**P**∨**R**))	DIS (distributività di ∨ rispetto a ∧)
¬(**P**∧**Q**)	(¬**P**∨¬**Q**)	DM (De Morgan)
¬(**P**∨**Q**)	(¬**P**∨¬**Q**)	DM (De Morgan)
(**P**→**Q**)	(¬**Q**→¬**P**)	CN (contrapposizione)
(**P**→**Q**)	(¬**P**∨**Q**)	EQ (equivalenza →/∨)
(**P**→**Q**)	¬(**P**∧¬**Q**)	EQ (equivalenza →/∧)
((**P**∧**Q**)→**R**)	(**P**→(**Q**→**R**))	ESP (esportazione)

Quando si applica una di queste regole di sostituzione in una derivazione, la giustificazione è costituita da 'RS' (regola di sostituzione), seguito dall'abbreviazione della regola applicata e dal numero della riga precedente contenente l'enunciato in cui si effettua la sostituzione.
Le seguenti brevi derivazioni illustrano l'applicazione corretta di alcune di queste regole.

Argomentazione 12

$(A\wedge(B\vee C))$
$\therefore (A\wedge(C\vee B))$

Derivazione

1. $(A\wedge(B\vee C))$: PREMESSA
2. $(A\wedge(C\vee B))$: RS CM,1

L'unica differenza fra la premessa e la conclusione è nel secondo congiunto, che nella premessa è '$(B\vee C)$', mentre nella conclusione è '$(C\vee B)$'. Poiché '$(B\vee C)$' è logicamente equivalente a '$(C\vee B)$', possiamo sostituire '$(B\vee C)$' con '$(C\vee B)$' nella premessa, ottenendo direttamente la conclusione.

Argomentazione 13

$(\neg A\rightarrow B)$
$\therefore (B\vee A)$

Derivazione

1. $(\neg A\rightarrow B)$: PREMESSA
2. $(\neg\neg A\vee B)$: RS EQ,1
3. $(A\vee B)$: RS DN,2
4. $(B\vee A)$: RS CM,3

Argomentazione 14

$(A\wedge\neg(B\rightarrow\neg C))$
$\therefore ((A\wedge B)\wedge C)$

Derivazione

1. $(A\wedge\neg(B\rightarrow\neg C))$: PREMESSA
2. $(A\wedge\neg(\neg B\vee\neg C))$: RS EQ,1
3. $(A\wedge(\neg\neg B\wedge\neg\neg C))$: RS DM,2
4. $(A\wedge(B\wedge\neg\neg C))$: RS DN,3
5. $(A\wedge(B\wedge C))$: RS DN,4
6. $((A\wedge B)\wedge C)$: RS AS,5

Argomentazione 15

$((A \lor \neg B) \to \neg E)$
$(A \lor (\neg B \land C))$
$\therefore \neg E$

Derivazione

1. $((A \lor \neg B) \to \neg E)$	: PREMESSA
2. $(A \lor (\neg B \land C))$	: PREMESSA
3. $((A \lor \neg B) \quad \neg (A \lor C))$	: RS DIS,2
4. $(A \lor \neg B)$	: $\neg$ELIM,3
5. $\neg E$	: $\to$ELIM,1,4

In tutte queste righe, la giustificazione fondamentale degli enunciati è "regola di sostituzione" (RS). Viene poi citato il motivo per cui l'enunciato sostitutivo è logicamente equivalente all'enunciato sostituito; tale motivo è solitamente costituito dall'abbreviazione di una delle coppie di enunciati logicamente equivalenti dell'elenco appena visto.
La regola RS conserva la verità? In altre parole: se partiamo da enunciati veri, deriveremo solo enunciati veri applicando, anche più volte, la regola RS? Il valore di verità di un enunciato composto è funzione dei valori di verità delle sue parti: questo fatto è stato ampiamente chiarito nei Capitoli dal 3 al 6; se dunque una di queste parti viene sostituita con un enunciato avente lo stesso valore di verità, il valore di verità dell'intero enunciato composto non risulterà cambiato. Se l'enunciato composto vale inizialmente TRUE, sostituendo una sua parte con un enunciato avente lo stesso valore di verità di tale parte si ottiene ancora un enunciato composto che vale TRUE. Ma gli enunciati logicamente equivalenti sono appunto enunciati che hanno lo stesso valore di verità in tutte le situazioni; perciò la regola RS conserva la verità: sostituendo una sottoformula di un enunciato composto con un enunciato logicamente equivalente ad essa, non si altera il valore di verità dell'enunciato composto.
A volte il nostro elenco di enunciati logicamente equivalenti può non essere disponibile, oppure può capitare di avere dei buoni motivi per ritenere logicamente equivalenti due enunciati non appartenenti a tale elenco, e di voler quindi applicare ad essi RS. Come ci si comporta in questi casi?
Si potrebbe dimostrare l'equivalenza logica di enunciati costruendo una tavola di verità in margine alla derivazione, per provare che i due enunciati hanno gli stessi valori di verità in tutte le situazioni. Questo metodo, però, se applicato a mano, richiederebbe spesso molto tempo e molto spazio; finiremmo poi col rendere ingarbugliata la nostra derivazione, altrimenti molto lineare, introducendovi una lunga digressione riguardante proprio quelle considerazioni sui valori di verità che una derivazione dovrebbe evitare.
Anziché costruire una tavola di verità, ricorreremo ad un criterio già visto: due enunciati **P** e **Q** sono logicamente equivalenti se e solo se il bicondizionale $(\mathbf{P} \leftrightarrow \mathbf{Q})$ è una tautologia. Inoltre, un enunciato è una tautologia se e solo se può essere

RESTITUITO da una porzione di derivazione che non applichi la regola INVIATO né a una premessa, né ad altre righe esterne a tale porzione. Infine, se si applica ↔INTR a due tautologie, il bicondizionale che si ottiene è una tautologia. In altre parole, si può dimostrare che un bicondizionale è una tautologia se si riesce a costruire una porzione di derivazione avente la seguente struttura:

/INIZIO: ↔INTR per derivare (**P**↔**Q**)/
  /INIZIO: →INTR per derivare (**P**→**Q**)/
  /FINE: →INTR per derivare (**P**→**Q**)/
  /INIZIO: →INTR per derivare (**Q**→**P**)/
  /FINE: →INTR per derivare (**Q**→**P**)/
/FINE: ↔INTR per derivare (**P**↔**Q**)/

e se in questa porzione nessuna riga contiene una giustificazione INVIATO che faccia riferimento a righe precedenti questa porzione stessa.
Si consideri l'argomentazione seguente:

Argomentazione 16

(B∧((A∨A)∧A))
∴(B∧A)

Cominciando con l'osservare che, se potessimo sostituire ((A∨A)∧A) con A, la derivazione sarebbe molto semplice. Appare inoltre logico supporre che gli enunciati

((A∨A)∧A)

e

A

siano logicamente equivalenti; in tal caso, la sostituzione sarebbe lecita, in base a RS. Si può procedere nel modo seguente:

1. (B∧((A∨A)∧A)) : PREMESSA
/INIZIO: ↔INTR per derivare (A↔((A∨A)∧A)) per RS/
/INIZIO: →INTR per derivare (A→((A∨A)∧A))/
*2. A : ASSUNZIONE
*3. (A∨A) : ∨INTR,2
*4. ((A∨A)∧A) : ∧INTR,3,2
*5. (A→((A∨A)∧A)) : →INTR,2,4
/FINE: →INTR per derivare (A→((A∨A)∧A))/
6. (A→((A∨A)∧A)) : RESTITUITO,5
/INIZIO: →INTR per derivare (((A∨A)∧A)→A)/

*7.	((A∨A)∧A)	: ASSUNZIONE
*8.	A	: ∧ELIM,7
*9.	(((A∨A)∧A)→A)	: →INTR,7,8
	/FINE: →INTR per derivare (((A∨A)∧A)→A)/	
10.	(((A∨A)∧A)→A)	: RESTITUITO,9
11.	(A↔((A∨A)∧A))	: ↔INTR,6,10
	/FINE: ↔INTR per derivare (A↔((A∨A)∧A))/	
12.	(B∧A)	: RS (2-11),1

Esaminando attentamente le righe dalla 2 alla 11 si può notare che in esse non è citata nessuna riga esterna a tale porzione di derivazione, la quale, dunque, essendo autosufficiente, può essere usata per dimostrare che l'enunciato della riga 11 è tautologia, e che quindi i due enunciati che lo compongono sono logicamente equivalenti. Alla riga 12 applichiamo la regola di sostituzione (RS), usando l'informazione racchiusa nelle righe dalla 2 alla 11, relativa al fatto che 'A' e '((A∨A)∧A)' sono logicamente equivalenti. La sostituzione di '((A∨A)∧A)' con 'A' è stata applicata alla riga 1, che quindi viene citata.

9.6 MODUS TOLLENS

Un'altra regola per l'eliminazione del condizionale è nota col nome latino di *modus tollens* (che abbrevieremo con MT); tale regola opera su enunciati contenenti → e ¬.

MT Da un enunciato del tipo
(P→Q)
e da uno del tipo
¬Q
si può derivare
¬P

In altre parole, se abbiamo un enunciato condizionale e la negazione del conseguente di tale enunciato, possiamo derivare la negazione dell'antecedente.
I ragionamenti basati sul *modus tollens* sono molto comuni nella vita di tutti i giorni. Ad esempio:

Se questo è un pino, deve avere le pigne.
Non ha le pigne.
Perciò non è un pino.

Per dimostrare che MT conserva la verità, si può derivare la conclusione desiderata:

1. **(P→Q)** : PREMESSA
2. ¬**Q** : PREMESSA
   /INIZIO: ¬INTR per derivare ¬**P**/
*3. **P** : ASSUNZIONE
*4. **(P→Q)** : INVIATO,1
*5. **Q** : →ELIM,4,3
*6. ¬**Q** : INVIATO,2
*7. ¬**P** : ¬INTR,3,5,6
   /FINE: ¬INTR per derivare ¬**P**/
8. ¬**P** : RESTITUITO,7

Poiché INVIATO, →ELIM, ¬INTR e RESTITUITO conservano la verità, la conserva anche MT. Abbiamo così dimostrato che quando si ha un enunciato del tipo **(P→Q)** e uno del tipo ¬**Q** si può derivare un enunciato del tipo ¬**P**.
Diremo che questa è una regola *derivata*, perché la sua validità si fonda su regole precedenti; in effetti, come avevamo già detto, tutte le regole presentate in questo capitolo sono derivabili dalle regole del Capitolo 8.
Dunque MT (o qualsiasi altra regola derivata), quando compare come giustificazione di una riga di una derivazione, può essere considerata l'abbreviazione della propria derivazione (sopra riportata). In una derivazione non abbreviata, cioè espressa per esteso, la riga avente MT come giustificazione viene sostituita da una copia della derivazione sopra riportata, adattata nel modo seguente: gli enunciati **P** e **Q** vengono sostituiti con gli enunciati cui MT si riferisce; le giustificazioni delle righe 1 e 2 vengono sostituite dalle effettive giustificazioni degli enunciati corrispondenti a **(P→Q)** e a ¬**Q**; i numeri di riga vengono opportunamente ridefiniti; e così via.
Vediamo un esempio per chiarire questo concetto:

Argomentazione 17

(¬A→B)
¬B
∴A

Derivazione abbreviata

1. (¬A→B) : PREMESSA
2. ¬B : PREMESSA
3. ¬¬A : MT,1,2
4. A : RS DN,3

Derivazione per esteso

1. (¬A→B) : PREMESSA.
2. ¬B : PREMESSA

*3. $\neg$A	: ASSUNZIONE
*4. ($\neg$A$\rightarrow$B)	: INVIATO,1
*5. B	: $\rightarrow$ELIM,4,3
*6. $\neg$B	: INVIATO,2
*7. $\neg\neg$A	: $\neg$INTR,3,5,6
8. $\neg\neg$A	: RESTITUITO,7 [al posto di MT,1,2]
9. A	: RS DN,8

Molte altre regole utili sono derivabili; alcune verranno esaminate negli esercizi. L'espansione di una derivazione abbreviata che usa una regola derivata corrisponde all'espansione delle "*macro*" in un linguaggio di programmazione assemblativo: anziché riportare in dettaglio una porzione di derivazione per derivare un enunciato in base a regole precedenti, attribuiamo un nome, come MT, ad una sequenza di passi ed usiamo tale nome per riferirci ad essa.

9.7 CONCLUSIONI

Per rendere più semplici e più naturali le derivazioni, abbiamo aggiunto al nostro sistema di deduzione naturale alcune regole di inferenza derivate, che permettono l'introduzione e l'eliminazione dei connettivi $\vee$, $\rightarrow$ e $\leftrightarrow$.

$\vee$INTR - Da **P** si può derivare (**P**$\vee$**Q**).
$\vee$ELIM - Da (**P**$\vee$**Q**) e $\neg$**P** si può derivare **Q**.
$\rightarrow$INTR - Se **P** è l'assunzione di una sottoprova e se **Q** è una riga di tale sottoprova, (**P**$\rightarrow$**Q**) è derivabile in quella sottoprova o può essere restituito.
$\rightarrow$ELIM - Da (**P**$\rightarrow$**Q**) e **P** si può derivare **Q**.
$\leftrightarrow$INTR - Da (**P**$\rightarrow$**Q**) e (**Q**$\rightarrow$**P**) si può derivare (**P**$\leftrightarrow$**Q**).
$\leftrightarrow$ELIM - Da (**P**$\leftrightarrow$**Q**) si può derivare o (**P**$\rightarrow$**Q**), o (**Q**$\rightarrow$**P**).

Abbiamo inoltre discusso una regola molto potente: la regola di sostituzione (RS); contestualmente, abbiamo elencato alcune equivalenze logiche di uso corrente:

P	$\neg\neg$**P**	DN (doppia negazione)
(**P**$\wedge$**Q**)	(**Q**$\wedge$**P**)	CM (commutatività di $\wedge$)
(**P**$\vee$**Q**)	(**Q**$\vee$**P**)	CM (commutatività di $\vee$)
P	(**P**$\wedge$**P**)	ID (idempotenza di $\wedge$)
P	(**P**$\vee$**P**)	ID (idempotenza di $\vee$)
(**P**$\wedge$(**Q**$\vee$**R**))	((**P**$\wedge$**Q**)$\wedge$**R**)	AS (associatività di $\wedge$)
(**P**$\vee$(**Q**$\vee$**R**))	((**P**$\vee$**Q**)$\vee$ **R**)	AS (associatività di $\vee$)
(**P**$\wedge$(**Q**$\vee$**R**))	((**P**$\wedge$**Q**)$\vee$(**P**$\wedge$**R**))	DIS (distributività di $\wedge$ rispetto a $\vee$)

$(P \vee (Q \wedge R))$	$((P \vee Q) \wedge (P \vee R))$	DIS (distributività di $\vee$ rispetto a $\wedge$)
$\neg(P \wedge Q)$	$(\neg P \vee \neg Q)$	DM (De Morgan)
$\neg(P \vee Q)$	$(\neg P \wedge \neg Q)$	DM (De Morgan)
$(P \rightarrow Q)$	$(\neg Q \rightarrow \neg P)$	CN (contrapposizione)
$(P \rightarrow Q)$	$(\neg P \vee Q)$	EQ (equivalenza $\rightarrow/\vee$)
$(P \rightarrow Q)$	$\neg(P \wedge \neg Q)$	EQ (equivalenza $\rightarrow/\wedge$)
$((P \wedge Q) \rightarrow R)$	$(P \rightarrow (Q \rightarrow R))$	ESP (esportazione)

Abbiamo infine introdotto un'ulteriore regola derivata:

MT (*modus tollens*) - Da $(P \rightarrow Q)$ e $\neg Q$ si può derivare $\neg P$.

9.8 ESERCIZI

A. Deduzioni naturali
Fornire una derivazione per ciascuna delle seguenti argomentazioni:

1. $(A \wedge B)$
   $(A \rightarrow C)$
   $\therefore C$

2. $(A \rightarrow B)$
   $(B \rightarrow C)$
   $(C \rightarrow D)$
   $\therefore (A \rightarrow D)$

3. $(A \rightarrow (B \vee C))$
   $(D \wedge A)$
   $\neg C$
   $\therefore B$

4. $(A \vee B)$
   $(A \vee \neg B)$
   $\therefore A$

5. A
   $\therefore (B \vee \neg B)$

6. $(A \vee B)$
   $\neg A$
   $(B \rightarrow C)$
   $\therefore C$

7. A
   $(A \rightarrow B)$
   $\neg B$
   $\therefore \neg A$

8. $(A \rightarrow (B \rightarrow C))$
   $(A \rightarrow B)$
   $\therefore (A \rightarrow C)$

9. $(\neg A \vee (B \wedge C))$
   $\neg B$
   $\therefore \neg A$

10. $(A \rightarrow B)$
    $\neg(B \vee C)$
    $\therefore \neg A$

B. Rappresentare le seguenti argomentazioni in forma simbolica e fornirne le derivazioni.

1. Il ministro dell'industria e gli industriali sono favorevoli alla legge, oppure sono favorevoli ad essa il ministro dell'industria e i sindacati. I sindacati sono favorevoli alla legge solo se la maggioranza degli iscritti è favorevole ad essa. Ma la maggioranza degli iscritti non è favorevole alla legge. Perciò gli industriali sono favorevoli alla legge.

2. Stasera la macchina la prenderà o mio figlio o mia figlia. Se mio figlio o mia figlia prenderà la macchina stasera, mia moglie non sarà contenta. Io sarò contento se nessuno dei due prenderà la macchina stasera. Ma se mia moglie non sarà contenta, io non sarò contento. E se io non sono contento, sono di cattivo umore. Perciò sarò di cattivo umore.
3. Il programma agonistico verrà abbandonato, a meno che non ci siano nuovi finanziamenti. Se verrà abbandonato, attireremo gli spettatori se e solo se siamo famosi. Ma noi attireremo gli spettatori e non siamo famosi. Perciò ci sono nuovi finanziamenti.
4. Se le scorte di petrolio si esauriranno il prezzo dell'elettricità aumenterà, e il costo dell'energia solare aumenterà purché le scorte di petrolio non si esauriscano. Se il prezzo dell'elettricità aumenterà, la gente avrà freddo. Perciò la gente non avrà freddo solo se il costo dell'energia solare aumenterà.
5. Il male non può esistere, a meno che Dio non sia privo della capacità o della volontà di impedirlo. Se Dio è onnipotente, è capace di impedire il male. Se Dio è infinitamente buono, ha la volontà di impedire il male. Se Dio esiste, è sia onnipotente, sia infinitamente buono. Tuttavia il male esiste. Perciò Dio non esiste.
6. Se le leggi sono giuste e vengono applicate con rigore, la criminalità diminuirà. Se l'applicazione rigorosa delle leggi comporta una diminuzione della criminalità, possiamo ridurre gli organici della polizia. Perciò, se non possiamo ridurre gli organici della polizia, le leggi non sono giuste.
7. Né Alberto né Carlo sono studiosi. Tuttavia Alberto è studioso se lo è Davide, oppure se Davide o Enrico sono studiosi allora Carlo è studioso se e solo se lo è Davide. Perciò Davide non è studioso.
8. Paolo è un filosofo solo se lo è Rodolfo oppure Carlo. Se Rodolfo è un filosofo, allora Alfredo e Ludovico sono filosofi. Se Alfredo è un filosofo, allora Ludovico lo è solo se lo è anche Ugo. Ma Ugo e Paolo non sono entrambi filosofi. Perciò Paolo è un filosofo solo se lo è anche Carlo.

C. La conclusione di una derivazione priva di premesse è detta *teorema logico*. Se le regole di derivazione sono corrette, un teorema è una tautologia. Per dimostrare un teorema si deve cominciare la derivazione con una o più assunzioni, che vengono poi eliminate con la regola RESTITUITO. Perciò l'ultima riga della derivazione appartiene alla prova principale ed è priva di asterischi. Dimostrare i seguenti teoremi:

1. $(A \rightarrow A)$
2. $((A \rightarrow B) \rightarrow ((B \rightarrow C) \rightarrow (A \rightarrow C)))$
3. $((A \rightarrow B) \rightarrow ((C \rightarrow A) \rightarrow (C \rightarrow B)))$
4. $((A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C)))$
5. $((A \rightarrow (B \rightarrow C)) \rightarrow (B \rightarrow (A \rightarrow C)))$

6. $((A \to (A \to B)) \to (A \to B))$
7. $((A \to B) \to (\neg B \to \neg A))$
8. $(A \to (\neg A \to B))$
9. $(\neg A \to (A \to B))$
10. $((\neg A \to A) \to A)$
11. $((A \to \neg A) \to \neg A)$
12. $(\neg (A \to B) \to A)$
13. $(\neg (A \to B) \to \neg B)$
14. $((A \wedge B) \leftrightarrow (B \wedge A))$
15. $((A \wedge (B \wedge C)) \leftrightarrow ((A \wedge B) \wedge C))$
16. $(((A \wedge B) \to C) \leftrightarrow (A \to (B \to C)))$
17. $((A \to (B \wedge C) \leftrightarrow ((A \to B) \wedge (A \to C)))$
18. $(((A \to B) \wedge (C \to D)) \to ((A \wedge C) \to (B \wedge D)))$
19. $(((\neg A \wedge B) \wedge A) \to B)$
20. $((A \to B) \leftrightarrow \neg (A \wedge \neg B))$
21. $((A \wedge B) \leftrightarrow \neg (A \to \neg B))$
22. $(\neg (A \wedge B) \leftrightarrow (A \to \neg B))$
23. $(A \leftrightarrow (A \wedge A))$
24. $((A \wedge \neg B) \to \neg (A \to B))$
25. $(\neg (A \wedge B) \leftrightarrow (\neg A \vee \neg B))$
26. $(\neg (A \vee B) \leftrightarrow (\neg A \wedge \neg B))$
27. $(((A \to B) \wedge (C \to B)) \leftrightarrow ((A \vee C) \to B))$
28. $((A \vee B) \leftrightarrow (B \vee A))$
29. $((A \vee (B \vee C)) \leftrightarrow ((A \vee B) \vee C))$
30. $((A \to (B \vee C)) \leftrightarrow ((A \to B) \vee (A \to C)))$
31. $((A \to B) \vee (B \to C))$
32. $(A \vee \neg A)$
33. $((A \wedge (B \vee C)) \leftrightarrow ((A \wedge B) \vee (A \wedge C)))$
34. $((A \vee (B \wedge C)) \leftrightarrow ((A \vee B) \wedge (A \vee C)))$
35. $(A \leftrightarrow ((A \wedge B) \vee (A \wedge \neg B)))$
36. $(A \leftrightarrow ((A \vee B) \wedge (A \vee \neg B)))$
37. $(A \leftrightarrow \neg \neg A)$
38. $((A \to B) \leftrightarrow (\neg A \vee B))$
39. $(((A \wedge B) \to C) \leftrightarrow (A \to (B \to C)))$
40. $((A \leftrightarrow B) \leftrightarrow ((A \to B) \wedge (B \to A)))$

D. Definire delle regole di introduzione e di eliminazione per NAND, NOR e XOR e dimostrare che conservano la verità.
E. Dimostrare che le regole seguenti sono derivabili:

1. Sillogismo ipotetico (SI):
   Da un condizionale del tipo
   $\mathbf{(P \to Q)}$
   e da uno del tipo
   $\mathbf{(Q \to R)}$

si può derivare il condizionale
$(P \rightarrow R)$

2. Dilemma costruttivo (DC)
Da una disgiunzione del tipo
$(P \rightarrow R)$
$(P \vee Q)$
e da due condizionali del tipo
$(Q \rightarrow S)$
si può derivare la disgiunzione
$(R \vee S)$

F. Dimostrare che le regole seguenti conservano la verità:

1. Sillogismo ipotetico (SI)
2. $\rightarrow$INTR

10

Logica enunciativa: un algoritmo per verificare prove

Nel Capitolo 5 abbiamo definito alcuni algoritmi per calcolare valori di verità e per determinare se un enunciato espresso in forma simbolica è ben formato. Nel Capitolo 6 abbiamo definito algoritmi per produrre una tavola di verità per un'argomentazione, allo scopo di determinare se tale argomentazione espressa in forma simbolica in logica enunciativa è valida oppure no.

Nei Capitoli 8 e 9 abbiamo definito le regole per dimostrare, tramite una derivazione, che un'argomentazione in logica enunciativa è valida. Questo metodo per illustrare perché un'argomentazione è valida è notevolmente meno gravoso del metodo delle tavole di verità dei Capitoli 5 e 6; esso riflette inoltre un modo più naturale di analizzare i motivi per cui un'argomentazione è valida. Occorre però tener presente che anche il metodo delle derivazioni presenta qualche punto debole. Se infatti un'argomentazione è valida, se ne può derivare la conclusione, evidenziando allo stesso tempo il motivo per cui è valida; se invece un'argomentazione non è valida, il metodo delle derivazioni non è utile: si finirebbe semplicemente col rimanere bloccati, senza riuscire a derivare la conclusione. Prima ancora di tentare di dimostrare la validità di un'argomentazione mediante una derivazione conviene perciò avere almeno un indizio a favore della sua validità.

Quando si applica il metodo delle derivazioni, descritto nei Capitoli 8 e 9, sorgono due problemi rilevanti:

1. Dopo aver scritto una derivazione, come si può essere certi che sia corretta, cioè che segua le regole che governano una derivazione?
2. Se siamo certi che un'argomentazione sia valida (eventualmente per averlo saputo da fonte attendibile), come possiamo generare una prova?

Il primo problema consiste nel determinare se una prova assegnata sia corretta; il secondo consiste invece nel generare la prova di un'argomentazione data.

Questi due problemi verranno trattati in questo capitolo e nel capitolo successivo.

Nel presente capitolo affronteremo il primo problema: come determinare se una

prova è corretta. A tal fine, definiremo un algoritmo, che chiameremo verifica-prova.
Nel Capitolo 11 affronteremo invece il secondo problema: come costruire una prova quando siano assegnate le premesse e la conclusione di un'argomentazione.

10.1 RIGHE DI UNA PROVA

L'algoritmo di verifica di una prova deve essere in grado di individuare qualunque errore che possa essere commesso: numeri di riga scorretti, regole usate impropriamente, sottoprove errate. L'algoritmo deve cioè essere in grado di rilevare diverse caratteristiche di ogni riga di una prova. Poiché una riga di derivazione contiene informazioni organizzate, può essere definita una *struttura di dati*: è anzi la struttura di dati fondamentale dell'algoritmo VERIFICA-PROVA.
Ogni riga di una prova deve avere la struttura seguente:

<asterischi><numero di riga>.<enunciato>:<regola>,<numeri di riga>

Ad esempio:

*4. (A∧B) : ∧INTR,2,3

In ogni riga esistono aree predefinite che devono contenere particolari informazioni; tali aree saranno dette *campi* di una riga.
Il primo campo contiene una stringa di asterischi, che può anche essere vuota (cioè priva di asterischi). Il numero di asterischi è pari al numero di assunzioni sotto cui si lavora in quella riga della prova. Questa parte di riga sarà detta *campo di profondità della sottoprova*, perché il numero di asterischi indica la "profondità" della sottoprova in quella riga.
Il secondo *campo* è quello *del numero di riga*, che indica semplicemente il numero d'ordine della riga nella prova e deve contenere un numero intero positivo (1, 2, 3, ...). Per comodità, il campo del numero di riga viene separato dal campo successivo mediante un punto ('.').
Il terzo campo deve contenere un enunciato ben formato, come '(A∧B)', '(C→D)', eccetera: è il *campo dell'enunciato*, separato dal campo successivo mediante due punti (':').
Il quarto campo contiene la giustificazione dell'enunciato contenuto appunto nel campo dell'enunciato, come PREMESSA, ∧INTR, ∨ELIM, eccetera. Questa parte di riga sarà chiamata *campo della regola*, in quanto il suo contenuto sarà definito come la *regola* usata in quella riga, anche se, in realtà, le due giustificazioni PREMESSA e ASSUNZIONE non sono regole secondo la terminologia fin qui usata. Se vi sono riferimenti a righe precedenti, il campo della regola è seguito da una virgola.

Infine, il quinto campo contiene eventualmente i numeri d'ordine di righe precedenti cui si debba far riferimento in base alla regola che si sia applicando. Le due regole PREMESSA e ASSUNZIONE non richiedono riferimenti a righe precedenti; in tal caso, perciò, l'ultimo campo, che chiameremo *campo dei riferimenti*, deve essere vuoto. Altrimenti, esso può contenere fino a tre numeri interi positivi, separati da virgole.
Si consideri, come esempio, la riga seguente:

***11. (A→(B∧C))   : →ELIM,2,4

Il contenuto dei suoi campi è:

Campo di profondità della sottoprova:	***
Campo del numero di riga:	11
Campo dell'enunciato:	(A→(B∧C))
Campo della regola:	→ELIM
Campo dei riferimenti:	2,4

Si noti che i campi di una riga sono identificabili in modo univoco; è cioè possibile scrivere un algoritmo per individuarli. Non definiremo qui tale algoritmo, perché dipende in misura rilevante dalla particolare struttura di dati scelta per rappresentare una riga; supporremo però che esso sia disponibile per essere usato in VERIFICA-PROVA.

10.2 L'ALGORITMO VERIFICA-PROVA

In una prova può comparire ogni tipo di errore; in ogni caso, però, ciascun errore comparirà in una riga della derivazione; in particolare:

Ci possono essere troppi, o troppo pochi, asterischi.
Il numero di riga può non essere corretto.
Il contenuto del campo dell'enunciato può non essere un enunciato ben formato.
Possono mancare il punto al termine del campo del numero di riga o i due punti al termine del campo dell'enunciato.
Il contenuto del campo della regola può non essere una regola corretta, oppure può essere una regola applicata in modo errato, che non giustifica l'enunciato contenuto nel campo dell'enunciato.
Infine, il contenuto del campo dei riferimenti può non essere appropriato per l'enunciato contenuto nel campo dell'enunciato e per la regola contenuta nel campo della regola.

Gli errori possibili, che devono essere identificati dall'algoritmo, sono insomma

parecchi. Divideremo dunque l'algoritmo VERIFICA-PROVA in diverse parti, o sottoprogrammi, ciascuno dei quali corrisponde ad una classe di errori.
I quattro sottoprogrammi di VERIFICA-PROVA sono:

VERIFICA-STRUTTURA-RIGA: controlla che ogni riga abbia il formato corretto e definisce i valori dei campi della riga corrente.
VERIFICA-STRUTTURA-ENUNCIATO: controlla che il campo dell'enunciato contenga un enunciato ben formato.
VERIFICA-REGOLA: controlla che il contenuto del campo della regola e il contenuto del campo dei riferimenti costituiscano una giustificazione corretta dell'enunciato contenuto nel campo dell'enunciato.
VERIFICA-SOTTOPROVA: controlla che ogni sottoprova abbia il numero appropriato di asterischi e segua le regole che governano le sottoprove.

L'algoritmo completo è il seguente:

ALGORITMO VERIFICA-PROVA

1. INPUT derivazione da verificare.
2. FOR ogni riga della derivazione
   (a) VERIFICA-STRUTTURA-RIGA.
   (b) VERIFICA-STRUTTURA-ENUNCIATO.
   (c) VERIFICA-REGOLA.
   (d) VERIFICA-SOTTOPROVA.
3. IF non ci sono errori
   THEN OUTPUT "Derivazione corretta, argomentazione valida."
4. STOP.

Raffiniamo ora i quattro sottoprogrammi.

IL SOTTOPROGRAMMA VERIFICA-STRUTTURA-RIGA

Cominciamo con il sottoprogramma più semplice e fondamentale: VERIFICA-STRUTTURA-RIGA. Come dev'essere esattamente la struttura di ogni riga di una prova? Ignoreremo le righe di commento, che cominciano con '/' e finiscono con '/', perché non fanno parte della prova, ma sono solo annotazioni personali. Ogni riga deve contenere un punto, che separa il campo del numero di riga dal campo dell'enunciato, e un segno di due punti, che separa il campo dell'enunciato dal campo della regola; fra l'inizio della riga e il punto deve comparire un numero; e così via. Queste sono però solo osservazioni sparse; definiremo ora un algoritmo per verificare se tutte le condizioni sono verificate.
Questo sottoprogramma, essendo ripetuto per ogni riga della derivazione, ad ogni chiamata riceve in ingresso una sola riga, che chiameremo RIGACORRENTE.

Questo identificatore ci permetterà di far riferimento ai contenuti dei campi di ciascuna riga. Ad esempio, PROFONDITÀ-SOTTOPROVA(RIGACORRENTE) sarà la profondità della sottoprova cui appartiene la riga corrente, mentre ENUNCIATO(RIGACORRENTE) sarà l'enunciato che compare nella riga corrente. Questo tipo di notazione può essere usato anche per far riferimento al contenuto di un campo di un'altra riga; ad esempio, ENUNCIATO(riga *n*) è il contenuto del campo dell'enunciato della riga *n*.

ALGORITMO VERIFICA-PROVA, SOTTOPROGRAMMA VERIFICA-STRUTTURA-RIGA

1. IF RIGACORRENTE non contiene punti
   THEN OUTPUT "Errore."
2. IF in RIGACORRENTE non compare un segno di due punti dopo il primo punto
   THEN OUTPUT "Errore."
3. LET PROFONDITÀ-SOTTOPROVA(RIGACORRENTE) = numero di asterischi contenuti nel campo di profondità della sottoprova di RIGACORRENTE.
4. IF il campo del numero di riga di RIGACORRENTE è vuoto
   THEN OUTPUT "Errore."
5. IF il campo del numero di riga di RIGACORRENTE non è vuoto
   THEN
   (a) LET NUMERORIGA(RIGACORRENTE) = numero contenuto nel campo del numero di riga di RIGACORRENTE.
   (b) IF NUMERORIGA(RIGACORRENTE) non è un numero intero positivo
       THEN OUTPUT "Errore."
   (c) IF questa è la prima chiamata del sottoprogramma VERIFICA-STRUTTURA-RIGA e NUMERO(RIGACORRENTE) $\neq 1$
       THEN OUTPUT "Errore."
   (d) IF questa non è la prima chiamata e NUMERORIGA(RIGACORRENTE) $\neq 1 +$ NUMERORIGA(riga precedente)
       THEN OUTPUT "Errore."
6. IF il campo dell'enunciato di RIGACORRENTE è vuoto
   THEN OUTPUT "Errore."
7. IF il campo dell'enunciato di RIGACORRENTE non è vuoto
   THEN LET ENUNCIATO(RIGACORRENTE) = enunciato contenuto nel campo dell'enunciato di RIGACORRENTE.
8. IF il campo della regola di RIGACORRENTE è vuoto
   THEN OUTPUT "Errore."
9. IF il campo della regola di RIGACORRENTE non è vuoto
   THEN LET REGOLA(RIGACORRENTE) = regola contenuta nel campo della regola di RIGACORRENTE.
10. IF il campo dei riferimenti di RIGACORRENTE non è vuoto
    THEN

(a) LET RIF1(RIGACORRENTE) = primo riferimento del campo.
(b) IF c'è un secondo riferimento
  THEN LET RIF2(RIGACORRENTE) = secondo riferimento.
(c) IF c'è un terzo riferimento
  THEN LET RIF3(RIGACORRENTE) = terzo riferimento.
(d) IF c'è un quarto riferimento
  THEN OUTPUT "Errore."
(e) IF qualche riferimento non è un numero intero positivo
  THEN OUTPUT "Errore."
(f) IF qualche riferimento è ⩾NUMERORIGA(RIGACORRENTE)
  THEN OUTPUT "Errore."

Fine del sottoprogramma VERIFICA-STRUTTURA-RIGA.
I passi di questo sottoprogramma controllano che tutti gli elementi fondamentali di una riga di derivazione siano a posto, prima che venga effettuato un esame più approfondito. I passi 3, 5(a), 7, 9 e da 10(a) a 10(c) definiscono anche i nomi dei campi della riga corrente, in modo che si possa far riferimento ad essi nel seguito.

IL SOTTOPROGRAMMA VERIFICA-STRUTTURA-ENUNCIATO

Per gli scopi che ci prefiggiamo in questo capitolo, il secondo sottoprogramma di VERIFICA-PROVA, cioè VERIFICA-STRUTTURA-ENUNCIATO, è molto semplice. Il primo sottoprogramma, cioè VERIFICA-STRUTTURA-RIGA, indica qual è la parte della riga che dovrebbe contenere la stringa ben formata; ma nel Capitolo 5 avevamo definito appunto un algoritmo per determinare se una stringa è ben formata: basta inserirlo qui. In altre parole, VERIFICA-STRUTTURA-ENUNCIATO è semplicemente l'algoritmo VERIFICA DI ENUNCIATI del Capitolo 5, che riceve in ingresso la stringa ENUNCIATO(RIGACORRENTE) e verifica se è ben formata.

IL SOTTOPROGRAMMA VERIFICA-REGOLA

Il terzo sottoprogramma, cioè VERIFICA-REGOLA, costituisce il cuore dell'algoritmo VERIFICA-PROVA, in quanto determina se la regola fornita come giustificazione di una riga è stata applicata correttamente. A causa del numero elevato di regole utilizzabili per giustificare un enunciato, questo sottoprogramma è piuttosto prolisso, giacché occorre esaminare le modalità di applicazione di ogni singola regola. Questo sottoprogramma riceve in ingresso REGOLA(RIGACORRENTE).

ALGORITMO VERIFICA-PROVA, SOTTOPROGRAMMA VERIFICA-REGOLA

0. IF REGOLA(RIGACORRENTE) = PREMESSA
  THEN

(a) IF PROFONDITÀ-SOTTOPROVA(RIGACORRENTE) ≠ 0
THEN OUTPUT "Errore."
(b) IF il campo dei riferimenti contiene uno o più numeri
THEN OUTPUT "Errore."

1. IF REGOLA(RIGACORRENTE) = ∧ELIM
THEN
(a) IF il campo dei riferimenti non contiene esattamente un numero
THEN OUTPUT "Errore."
[perché la regola ∧ELIM fa riferimento a una sola riga precedente]
(b) Indicando con ENUNCIATO(RIF1(RIGACORRENTE)) l'enunciato contenuto nella riga cui fa riferimento RIF1(RIGACORRENTE)
IF ENUNCIATO(RIF1(RIGACORRENTE)) ≠ (ENUNCIATO(RIGACORRENTE) ∧**P**) oppure ≠ (**P**∧ENUNCIATO(RIGACORRENTE)), dove **P** è un enunciato qualsiasi
THEN OUTPUT "Errore."
[la riga precedente cui si fa riferimento deve cioè essere una congiunzione, ossia un enunciato avente ∧ come connettivo principale, di cui l'enunciato inferito, cioè ENUNCIATO(RIGACORRENTE), deve essere un congiunto]

2. IF REGOLA(RIGACORRENTE) = ∧INTR
THEN
(a) IF il campo dei riferimenti non contiene esattamente due numeri
THEN OUTPUT "Errore."
(b) IF ENUNCIATO(RIGACORRENTE) ≠ (ENUNCIATO(RIF1(RIGACORRENTE))∧ENUNCIATO(RIF2 (RIGACORRENTE)))
THEN OUTPUT "Errore."

Il passo 2 richiede che la regola citi due e due soli numeri di riga precedenti. Il passo 2(b) richiede che l'enunciato di RIGACORRENTE sia costituito dalla sequenza: '('; enunciato della prima riga citata; simbolo '∧'; enunciato della seconda riga citata; ')'.
La regola ∧INTR è stata infatti definita in modo molto preciso. Sia data, ad esempio, la prova seguente:

1. A : PREMESSA
2. B : PREMESSA
3. (A∧B) : ∧INTR,2,1 [errato]

La riga 3 contiene un errore, benché di scarso rilievo, in quanto cita la riga 2 prima della riga 1. Ciò significa che l'enunciato della riga 2 deve essere il primo congiunto dell'enunciato della riga 3, mentre l'enunciato della riga 2 deve esserne il secondo congiunto; tale condizione non è però verificata in questo esempio.
La riga 3 può essere corretta in due modi, sempre mantenendo ∧INTR come giu-

stificazione; potremmo lasciare invariati la regola e i riferimenti, correggendo l'enunciato:

3. (B∧A) : ∧INTR,2,1 [corretto]

oppure potremmo cambiare i riferimenti e mantenere inalterato l'enunciato:

3. (A∧B) : ∧INTR,1,2 [corretto]

Questo esempio mostra che si devono usare le regole con estrema attenzione, se si vuole sottoporle ad un algoritmo. Potremmo, in effetti, ammettere una maggiore libertà nell'uso dei riferimenti, ma in tal caso l'algoritmo sarebbe più complicato. Ad esempio, se si ammettesse di poter scrivere i riferimenti in un ordine qualsiasi, si dovrebbe modificare il passo 2(b) nel modo seguente:

```
2(b) [sperimentale]
     IF ENUNCIATO(RIGACORRENTE) ≠
     (ENUNCIATO(RIF1(RIGACORRENTE))∧
     ENUNCIATO(RIF2(RIGACORRENTE))) e ENUNCIATO
     (RIGACORRENTE) ≠
     (ENUNCIATO(RIF2(RIGACORRENTE))∧
     ENUNCIATO(RIF1(RIGACORRENTE)))
       THEN OUTPUT "Errore."
```

Questo esempio evidenzia uno spiacevole compromesso cui si deve spesso sottostare quando si scrivono degli algoritmi: per rendere un algoritmo comodo e versatile per un utente umano, lo si deve complicare; viceversa, per avere un algoritmo semplice, si devono adottare regole definite rigidamente e non sempre comode per un essere umano.
Continuiamo con il sottoprogramma VERIFICA-REGOLA:

```
3. IF REGOLA(RIGACORRENTE)=→ELIM
   THEN
   (a) IF il campo dei riferimenti non contiene esattamente due numeri
         THEN OUTPUT "Errore."
   (b) IF ENUNCIATO(RIF1(RIGACORRENTE)) ≠
       (ENUNCIATO(RIF2(RIGACORRENTE))→ENUNCIATO
       (RIGACORRENTE))
         THEN OUTPUT "Errore."
```

Il passo 3(a) richiede che la regola →ELIM faccia riferimento a due, e due sole, righe precedenti. Il passo 3(b) richiede che l'enunciato della seconda riga citata sia l'antecedente dell'enunciato della prima riga citata e che l'enunciato di RIGACORRENTE sia il conseguente dell'enunciato della prima riga citata. Ovviamente, affinché ciò sia verificato, occorre che l'enunciato della prima riga citata sia

un condizionale, cioè un enunciato avente → come connettivo principale. Anche questa regola è alquanto rigida, poiché i numeri di riga citati devono comparire nel giusto ordine: prima il condizionale, poi l'antecedente. Ad esempio, in:

1. (A→B) : PREMESSA
2. A : PREMESSA
3. B : →ELIM,2,1 [errato]

la riga 3 è scorretta, e provoca quindi un messaggio di "Errore", perché l'enunciato della riga 2 (che è la prima riga citata nella riga 3) non è identico a:

(ENUNCIATO(riga 1)→ENUNCIATO(riga 3))

Questo errore può essere corretto invertendo l'ordine dei numeri di riga citati nella riga 3:

3. B : ELIM,1,2 [corretto]

Questa riga è corretta, e non provoca un messaggio di "Errore", perché in questo modo la condizione

ENUNCIATO(riga 1) = (ENUNCIATO(riga 2)→ENUNCIATO(riga 3))

risulta verificata.
Il passo successivo del sottoprogramma VERIFICA-REGOLA è:

4. IF REGOLA(RIGACORRENTE) = →INTR
   THEN
   (a) IF il campo dei riferimenti non contiene esattamente due numeri
       THEN OUTPUT "Errore."
   (b) IF RIF2(RIGACORRENTE) < RIF1(RIGACORRENTE)
       THEN OUTPUT "Errore."
   (c) IF REGOLA(RIF1(RIGACORRENTE)) ≠ ASSUNZIONE
       THEN OUTPUT "Errore."
   (d) IF ENUNCIATO(RIGACORRENTE) ≠
       (ENUNCIATO(RIF1(RIGACORRENTE))→
       ENUNCIATO(RIF2 (RIGACORRENTE))
       THEN OUTPUT "Errore."

Questo passo controlla l'uso della regola →INTR. Ulteriori condizioni per la sua corretta applicazione saranno discusse nel paragrafo relativo a VERIFICA-SOTTOPROVA, perché →INTR comporta il ricorso a una sottoprova.
Il passo 4(a) richiede che si faccia riferimento esattamente a due righe. Il passo 4(b) richiede che il conseguente del condizionale derivato appartenga ad una riga successiva all'assunzione. Il passo 4(c) richiede che la regola della prima riga cita-

ta sia ASSUNZIONE (è qui che deve cominciare una sottoprova e deve essere introdotto un asterisco, se la giustificazione è ASSUNZIONE; ma questo argomento verrà discusso più avanti). Il passo 4(d) richiede che in RIGACORRENTE venga derivato un condizionale avente una determinata struttura.
Continuiamo con il sottoprogramma:

```
5. IF REGOLA(RIGACORRENTE) = ¬ELIM
     THEN
     (a) IF il campo dei riferimenti non contiene esattamente tre numeri
           THEN OUTPUT "Errore."
     (b) IF RIF2 o RIF3 < RIF1
           THEN OUTPUT "Errore."
     (c) IF ENUNCIATO(RIF1(RIGACORRENTE)) ≠
         negazione di ENUNCIATO(RIGACORRENTE)
           THEN OUTPUT "Errore."
     (d) IF REGOLA(RIF1(RIGACORRENTE)) ≠ ASSUNZIONE
           THEN OUTPUT "Errore."
     (e) IF ENUNCIATO(RIF3(RIGACORRENTE)) ≠
         negazione di ENUNCIATO(RIF2(RIGACORRENTE))
           THEN OUTPUT "Errore."
6. IF REGOLA(RIGACORRENTE) = ¬INTR
     THEN
     (a) IF il campo dei riferimenti non contiene esattamente tre numeri
           THEN OUTPUT "Errore."
     (b) IF RIF2 o RIF3 < RIF1
             THEN OUTPUT "Errore."
     (c) IF ENUNCIATO(RIF3(RIGACORRENTE)) ≠
         negazione di ENUNCIATO(RIF2(RIGACORRENTE))
           THEN OUTPUT "Errore."
     (d) IF REGOLA(RIF1(RIGACORRENTE)) ≠ ASSUNZIONE
           THEN OUTPUT "Errore."
     (e) IF ENUNCIATO(RIF3(RIGACORRENTE)) ≠
         negazione di ENUNCIATO(RIF2(RIGACORRENTE))
           THEN OUTPUT "Errore."
```

I passi che controllano $\neg$INTR e $\neg$ELIM sono identici, eccettuato (c): nel caso di $\neg$INTR, l'enunciato di RIGACORRENTE deve essere la negazione dell'enunciato della prima riga citata, mentre nel caso di $\neg$ELIM l'enunciato della prima riga citata deve essere la negazione dell'enunciato di RIGACORRENTE.
Il passo (a) richiede che la regola faccia riferimento a tre, e tre sole, righe. Il passo (d) richiede che la regola si fondi su un'ASSUNZIONE. Il passo (e) richiede che si faccia riferimento a due righe contenenti enunciati contraddittori, cioè che non possano valere TRUE contemporaneamente. Il passo (b) verifica che questi enunciati contraddittori compaiano, nella derivazione, dopo l'ASSUNZIONE.

I cinque passi successivi del sottoprogramma sono:

7. IF REGOLA(RIGACORRENTE) = ∨ELIM
   THEN
   (a) IF il campo dei riferimenti non contiene esattamente due numeri
       THEN OUTPUT "Errore."
   (b) IF ENUNCIATO(RIF1(RIGACORRENTE)) non è una disgiunzione del tipo
       (i) (**P**∨ENUNCIATO(RIGACORRENTE)) oppure
       (ii) (ENUNCIATO(RIGACORRENTE)∨**Q**)
       THEN OUTPUT "Errore."
   (c) IF ENUNCIATO(RIF1(RIGACORRENTE)) è del tipo (i) e ENUNCIATO(RIF2(RIGACORRENTE)) non è del tipo ¬**P** dove **P** è il primo disgiunto di ENUNCIATO(RIF1(RIGACORRENTE))
       THEN OUTPUT "Errore."
   (d) IF ENUNCIATO(RIF1(RIGACORRENTE)) è del tipo (ii) e ENUNCIATO(RIF2(RIGACORRENTE)) non è del tipo ¬**Q** dove **Q** è il secondo disgiunto di ENUNCIATO(RIF1(RIGACORRENTE))
       THEN OUTPUT "Errore."

8. IF REGOLA(RIGACORRENTE) = ∨INTR
   THEN
   (a) IF il campo dei riferimenti non contiene esattamente un numero
       THEN OUTPUT "Errore."
   (b) IF ENUNCIATO(RIGACORRENTE) ≠ (ENUNCIATO(RIF1(RIGACORRENTE))∨**P**)
       e ENUNCIATO(RIGACORRENTE) ≠ (**P**∨ENUNCIATO(RIF1(RIGACORRENTE)))
       dove **P** è un enunciato ben formato
       THEN OUTPUT "Errore."
9. IF REGOLA(RIGACORRENTE) = ↔ELIM
   THEN
   (a) IF il campo dei riferimenti non contiene esattamente un numero
       THEN OUTPUT "Errore."
   (b) IF ENUNCIATO(RIF1(RIGACORRENTE)) non è un bicondizionale del tipo (**P**↔**Q**)
       THEN OUTPUT "Errore."
   (c) IF ENUNCIATO(RIGACORRENTE) ≠ (**P**→**Q**)
       e ENUNCIATO(RIGACORRENTE) ≠ (**Q**→**P**)
       THEN OUTPUT "Errore."
10. IF REGOLA(RIGACORRENTE) = ↔INTR
    THEN
    (a) IF il campo dei riferimenti non contiene esattamente due numeri
        THEN OUTPUT "Errore."

(b) IF ENUNCIATO(RIF1(RIGACORRENTE)) e ENUNCIATO(RIF2(RIGACORRENTE)) non sono condizionali rispettivamente del tipo

(P→Q)

e

(Q→P)

dove **P** e **Q** sono enunciati ben formati tali che ENUNCIATO(RIGACORRENTE) = **(P↔Q)**
THEN OUTPUT "Errore."

11. IF REGOLA(RIGACORRENTE) non è né una di quelle precedentemente trattate, né ASSUNZIONE, INVIATO o RESTITUITO
THEN OUTPUT "Errore."

Il passo 11, che è l'ultimo di VERIFICA-REGOLA, richiede che la regola sia una di quelle discusse nei Capitoli 8 e 9. Le regole ASSUNZIONE, INVIATO e RESTITUITO saranno controllate dal prossimo sottoprogramma, VERIFICA-SOTTOPROVA.

IL SOTTOPROGRAMMA VERIFICA-SOTTOPROVA

Il sottoprogramma VERIFICA-SOTTOPROVA (l'ultimo di VERIFICA-PROVA) controlla la correttezza di certe regole che riguardano le sottoprove.

ALGORITMO VERIFICA-PROVA, SOTTOPROGRAMMA VERIFICA-SOTTOPROVA

1. IF PROFONDITÀ-SOTTOPROVA(riga 1) > 1
THEN OUTPUT "Errore."
2. IF PROFONDITÀ-SOTTOPROVA(riga1) = 1
e REGOLA(riga 1) ≠ ASSUNZIONE
THEN OUTPUT "Errore".
3. IF fra PROFONDITÀ-SOTTOPROVA(RIGACORRENTE) e PROFONDITÀ-SOTTOPROVA(riga precedente) c'è una differenza > 1
THEN OUTPUT "Errore."
4. IF PROFONDITÀ-SOTTOPROVA(RIGACORRENTE) =
1 + PROFONDITÀ-SOTTOPROVA(riga precedente)
e REGOLA(RIGACORRENTE) ≠ ASSUNZIONE
THEN OUTPUT "Errore".
5. IF PROFONDITÀ-SOTTOPROVA(RIGACORRENTE) =
PROFONDITÀ-SOTTOPROVA(riga precedente) − 1
e REGOLA(RIGACORRENTE) ≠ RESTITUITO
THEN OUTPUT "Errore".
6. IF REGOLA(RIGACORRENTE) = ASSUNZIONE
THEN

(a) IF PROFONDITÀ-SOTTOPROVA(RIGACORRENTE) ≠ 1 + PROFONDITÀ-SOTTOPROVA(riga precedente)
THEN OUTPUT "Errore".
(b) IF il campo dei riferimenti contiene uno o più numeri
THEN OUTPUT "Errore".

7. IF REGOLA(RIGACORRENTE) = INVIATO
THEN
(a) IF il campo dei riferimenti non contiene esattamente un numero
THEN OUTPUT "Errore."
(b) IF ENUNCIATO(RIGACORRENTE) ≠ ENUNCIATO(RIF1(RIGACORRENTE))
THEN OUTPUT "Errore."
(c) IF PROFONDITÀ-SOTTOPROVA(RIF1(RIGACORRENTE)) non è minore di PROFONDITÀ-SOTTOPROVA(RIGACORRENTE)
THEN OUTPUT "Errore."
(d) IF esiste un numero di riga *n* maggiore di RIF1(RIGACORRENTE) e minore di NUMERORIGA(RIGACORRENTE) tale che PROFONDITÀ-SOTTOPROVA(*n*) sia minore di PROFONDITÀ-SOTTOPROVA(RIF1(RIGACORRENTE))
THEN OUTPUT "Errore".

8. IF REGOLA(RIGACORRENTE) = RESTITUITO
THEN
(a) IF il campo dei riferimenti non contiene esattamente un numero
THEN OUTPUT "Errore".
(b) IF REGOLA(RIF1(RIGACORRENTE)) non è →INTR, ¬INTR o ¬ELIM
THEN OUTPUT "Errore."
(c) IF PROFONDITÀ-SOTTOPROVA(RIF1(RIGACORRENTE)) ≠ PROFONDITÀ-SOTTOPROVA(RIGACORRENTE) + 1
THEN OUTPUT "Errore."
(d) IF ENUNCIATO(RIGACORRENTE) ≠ ENUNCIATO(RIF1(RIGACORRENTE))
THEN OUTPUT "Errore."
(e) IF NUMERORIGA(RIGACORRENTE) ≠ RIF1(RIGACORRENTE) + 1
THEN OUTPUT "Errore."

Il passo 1 richiede che la prima riga non abbia più di un asterisco. Il passo 2 richiede che la prima riga, se ha un asterisco, abbia la giustificazione ASSUNZIONE, cioè sia l'inizio di una sottoprova. I primi due passi si distinguono dai passi successivi perché riguardano la prima riga della derivazione, e non RIGACORRENTE; ma su questo argomento torneremo più avanti.
Il passo 3 controlla che il numero di asterischi di una riga differisca, al più, di un asterisco rispetto alla riga precedente. Il passo 4 controlla che una riga possa

aumentare il numero di asterischi (cioè la profondità di sottoprova) rispetto alla riga precedente solo usando la regola ASSUNZIONE. Il passo 5 controlla che una riga possa ridurre il numero di asterischi (cioè la profondità di sottoprova) rispetto alla riga precedente solo usando la regola RESTITUITO.
La seguente prova, piena di errori, illustra le applicazioni di queste condizioni:

1. **P	:PREMESSA	[errore al passo 1]
***2. **Q**	:ASSUNZIONE	
3. **(P∧Q)	:∧INTR,1,2	[errore al passo 5]
***4. **((P∧Q)∧Q)**	:∧INTR,3,2	[errore al passo 4]
5. **(P∧Q)	:∧ELIM,4	[errore al passo 5]
6. **P**	:∧ELIM,5	[errore al passo 3]

I passi da 6 a 8 controllano l'applicazione delle giustificazioni ASSUNZIONE, INVIATO e RESTITUITO, che sono trattate da VERIFICA-SOTTOPROVA, anziché da VERIFICA-REGOLA, perché vengono usate solo in relazione a sottoprove. INVIATO e RESTITUITO, come si è visto nei capitoli precedenti, hanno rispettivamente lo scopo di inviare informazioni in una sottoprova e di recuperare informazioni ottenute in una sottoprova.
Se una riga ha la giustificazione INVIATO, deve far riferimento ad una e una sola riga precedente, la quale deve contenere un enunciato identico a quello di RIGA-CORRENTE, poiché INVIATO copia semplicemente tale enunciato. I passi 7(c) e 7(d) hanno una struttura complicata, ma si fondono su un concetto semplice: le informazioni contenute in una sottoprova di profondità minore possono essere inviate in una sottoprova di profondità maggiore, ma non viceversa. Ad esempio, si consideri la seguente porzione di prova, che è corretta:

1. **P**	: PREMESSA
*2. **Q**	: ASSUNZIONE
*3. **P**	: INVIATO,1
*4. **(Q→P)**	: →INTR,2,3

In questo caso, l'informazione costituita dalla premessa **P** viene semplicemente ricopiata nella riga 3 per mezzo della regola INVIATO. La profondità di sottoprova della riga 1 è 0, quindi è minore della profondità di sottoprova della riga 3, che vale 1. Poiché **P** è dato come premessa, la sua ripetizione non può avere conseguenze negative.
Si consideri invece la seguente porzione di prova, che è scorretta:

1. **Q**	: PREMESSA
*2. **P**	: ASSUNZIONE
*3. **Q**	: INVIATO,1
*4. **(P→Q)**	:→INTR,2,3
5. **(P→Q)**	: RESTITUITO,4
6. **P**	: INVIATO,2 [errore al passo 7(c)]

La riga 6 è errata perché le informazioni contenute in una sottoprova non possono essere inviate ad una riga avente profondità di sottoprova minore, per lo meno se la giustificazione è INVIATO. In effetti, non esiste alcun motivo per credere che **P** sia vero, salvo quando lo usiamo come assunzione nella sottoprova che si estende dalla riga 2 alla riga 4; ma nella riga 6 siamo fuori da questa sottoprova. Perciò nel contesto della riga 6 non esiste alcuna giustificazione che permetta di asserire **P**.

Il passo 8 specifica le restrizioni sull'uso della regola RESTITUITO. Come si è già detto, RESTITUITO estrae informazioni da una sottoprova, affinché esse vengano usate in una porzione della prova che si trovi ad una profondità di sottoprova minore. Questa regola restituisce informazioni all'esterno di una sottoprova, anziché inviarne al suo interno. Come indica il passo 8(b), questa regola può essere usata solo quando la riga cui fa riferimento contiene la giustificazione →INTR, ¬INTR o ¬ELIM. Il passo 8(c) controlla che RESTITUITO trasferisca informazioni da una profondità di sottoprova maggiore a una minore. Il passo 8(d) richiede che una riga avente RESTITUITO come giustificazione si limiti a ripetere, cioè a copiare, l'enunciato della riga cui fa riferimento. Il passo 8(e) controlla che una nuova sottoprova abbia una profondità di sottoprova maggiore della precedente.

Infine, l'ultimo passo di VERIFICA-SOTTOPROVA controlla che le informazioni possano essere estratte da una sottoprova oppure inserite in essa soltanto per mezzo delle regole RESTITUITO e INVIATO:

```
9. FOR ogni riga L
   (a) IF PROFONDITÀ-SOTTOPROVA(RIF1(L)) o
       PROFONDITÀ-SOTTOPROVA(RIF2(L)) o
       PROFONDITÀ-SOTTOPROVA(RIF3(L))
       <PROFONDITÀ-SOTTOPROVA(RIGACORRENTE)
       e REGOLA(RIGACORRENTE)≠INVIATO
         THEN OUTPUT "Errore."
   (b) IF PROFONDITÀ-SOTTOPROVA(RIF1(L)) o
       PROFONDITÀ-SOTTOPROVA(RIF2(L)) o
       PROFONDITÀ-SOTTOPROVA(RIF3(L))
       >PROFONDITÀ-SOTTOPROVA(RIGACORRENTE)
       e REGOLA(RIGACORRENTE)≠RESTITUITO
         THEN OUTPUT "Errore."
   (c) FOR ogni riferimento contenuto in L
         IF esiste una riga intermedia n tale che
         PROFONDITÀ-SOTTOPROVA(n)
         <PROFONDITÀ-SOTTOPROVA(RIGACORRENTE)
         e REGOLA(RIGACORRENTE)≠INVIATO
           THEN OUTPUT "Errore."
```

Questo passo conclude l'ultimo dei quattro sottoprogrammi dell'algoritmo VERIFICA-PROVA.

10.3 APPLICAZIONI E MODIFICHE DI VERIFICA-PROVA

Riportiamo nuovamente l'algoritmo completo:

ALGORITMO VERIFICA-PROVA

1. INPUT derivazione da verificare.
2. FOR ogni riga della derivazione
   (a) VERIFICA-STRUTTURA-RIGA.
   (b) VERIFICA-STRUTTURA-ENUNCIATO.
   (c) VERIFICA-REGOLA.
   (d) VERIFICA-SOTTOPROVA.
3. IF non ci sono errori
   THEN OUTPUT "Derivazione corretta, argomentazione valida."
4. STOP.

Quando si applica questo algoritmo, vengono ripetuti inutilmente più volte i primi due passi del sottoprogramma VERIFICA-SOTTOPROVA. Poiché, come si ricorderà, tali passi riguardano soltanto la prima riga della prova, l'esame della prima riga verrebbe inutilmente ripetuto al momento della verifica di ciascuna riga della prova. Per rendere più efficiente l'algoritmo, lo si può modificare nel modo seguente:

ALGORITMO VERIFICA-PROVA-1

1. INPUT derivazione da verificare.
2. Eseguire i passi 1 e 2 del sottoprogramma VERIFICA-SOTTOPROVA.
3. FOR ogni riga della derivazione
   (a) VERIFICA-STRUTTURA-RIGA.
   (b) VERIFICA-STRUTTURA-ENUNCIATO.
   (c) VERIFICA-REGOLA.
   (d) VERIFICA-SOTTOPROVA, passi dal 3 al 9.
4. IF non ci sono errori
   THEN OUTPUT "Derivazione corretta, argomentazione valida."
5. STOP.

Se vogliamo applicare l'algoritmo personalmente, possiamo usarlo per diversi scopi. Potremmo applicarlo ad ogni riga di una prova che abbiamo scritto: se lo eseguiamo correttamente, senza mai generare un messaggio di errore, siamo certi che la prova sia corretta.
In altri casi, potremmo aver bisogno di verificare la correttezza di una sola riga, se siamo praticamente sicuri che le altre siano corrette: modificare l'algoritmo per

applicarlo ad una sola riga è un'operazione semplice e veloce.
Infine, se stiamo correggendo una prova per evidenziare gli errori che essa contiene, potremmo descrivere i tipi di errore riscontrati citando i corrispondenti passi dell'algoritmo; ad esempio, potremmo scrivere, al termine di una riga errata: "Errore rilevato al passo 4(b) di VERIFICA-REGOLA".
Consideriamo come esempio la seguente prova, che contiene diversi errori, alcuni dei quali evidenziati nel modo appena descritto:

1. $\neg A$	: PREMESSA	
2. $(B \rightarrow (A \wedge C))$	: PREMESSA	
*3. B	: ASSUNZIONE	
*4. $\neg A$	: INVIATO,1	
*5. $(A \wedge C)$	: $\rightarrow$ELIM,3,2	[errore al passo 3(b) di VERIFICA-REGOLA e al passo 8(b) di VERIFICA-SOTTOPROVA]
*6 A	: $\wedge$ELIM,5	
*7. $\neg B$	: $\neg$INTR,3,4,6	[errore al passo 6(c) di VERIFICA-REGOLA]
*8. $\neg B$	: RESTITUITO,7	

Correggendo la prova si ottiene:

1. $\neg A$	: PREMESSA
2. $(B \rightarrow (A \wedge C))$	: PREMESSA
*3. B	: ASSUNZIONE
*4. $\neg A$	: INVIATO,1
*5. $(B \rightarrow (A \wedge C))$	: INVIATO,2
*6. $(A \wedge C)$	: $\rightarrow$ELIM,5,3
*7. A	: $\wedge$ELIM,6
*8. $\neg B$	: $\neg$INTR,3,7,4
9. $\neg B$	: RESTITUITO,8

In questo algoritmo si possono introdurre diversi miglioramenti. Un difetto consiste nel fatto che, quando viene rilevato un errore, viene emesso soltanto il messaggio "Errore", ma non vengono specificati né il numero della riga in cui l'errore compare, né il tipo di errore individuato. Tale messaggio non fornisce dunque indicazioni utili per la correzione di una prova, poiché si limita ad avvertire che essa è errata in qualche punto.
Ad esempio, se la prova errata appena vista fosse fornita in ingresso ad un calcolatore programmato per eseguire l'algoritmo VERIFICA-PROVA, esso stamperebbe i messaggi:

Errore
Errore
Errore

e nient'altro. Sarebbe quindi assai difficile capire che cosa si deve correggere nella prova, perché sapremmo solo che essa contiene tre errori da qualche parte. Quando invece applichiamo a mano l'algoritmo, vediamo subito dove si trova ciascun errore.
Se si vuole tradurre l'algoritmo VERIFICA-PROVA in un effettivo programma per calcolatore è senz'altro necessario effettuare alcune aggiunte, a meno che non ci si accontenti semplicemente di contare gli errori. Probabilmente, le due aggiunte più significative sono le seguenti:

A. In ogni gruppo dell'algoritmo VERIFICA-PROVA in cui compare
   OUTPUT "Errore"
   si sostituisca tale istruzione con
   OUTPUT "Errore alla riga" NUMERORIGA(RIGACORRENTE).
B. Ad ogni istruzione che emette un messaggio di errore si aggiunga qualche chiarimento sulla natura dell'errore stesso, come:
   Enunciato non ben formato.
   La seconda riga citata non contiene la negazione della terza riga citata.
   L'enunciato non è una congiunzione del tipo giusto.
   Regola non nota.
   La regola $\wedge$INTR deve far riferimento a due righe.
   e così via.

10.4 CONCLUSIONI

In questo capitolo è stato illustrato l'algoritmo VERIFICA-PROVA, che riceve in ingresso quella che dovrebbe essere una derivazione nel sistema di deduzione naturale definito nei Capitoli 8 e 9 e fornisce in uscita o dei messaggi "Errore", se l'ingresso non è conforme alle regole per una prova corretta, o il messaggio "Derivazione corretta, argomentazione valida" se l'ingresso è conforme a tutte le regole.
VERIFICA-PROVA esamina la struttura di ogni riga della derivazione, poiché ogni errore si può verificare solo nella struttura di una riga. Una riga è costituita, nell'ordine, da: zero o più asterischi (zero per le righe della prova principale, uno o più per righe appartenenti a sottoprove); un numero intero; un punto; un enunciato ben formato; due punti (:); una regola del sistema di deduzione naturale; una virgola; da zero a tre numeri di riga, usati come riferimenti della regola.
Ci sono quattro sottoprogrammi: VERIFICA-STRUTTURA-RIGA controlla che ogni riga abbia il formato corretto; se una riga non ha il formato corretto, la derivazione contiene un errore sintattico. VERIFICA-STRUTTURA-ENUNCIATO controlla che l'enunciato sia ben formato; se non lo è, la derivazione contiene un errore sintattico. VERIFICA-REGOLA controlla che la regola e i relativi riferimenti giustifichino correttamente l'enunciato; se ciò non avviene, la derivazione

è scorretta. VERIFICA-SOTTOPROVA controlla che ogni sottoprova sia scritta correttamente; se esistono sottoprove scorrette, la derivazione contiene un errore sintattico.

10.5 ESERCIZI

A. Applicare VERIFICA-PROVA per individuare gli errori delle derivazioni seguenti.

1. (a)

1.	$(A \rightarrow B)$	: PREMESSA
2.	$(C \wedge \neg B)$	: ASSUNZIONE
3.	$\neg B$	: $\wedge$ELIM,2
*4.	A	: PREMESSA
*5.	B	: $\rightarrow$ELIM,4,1
6.	$\neg B$	: INVIATO,3
*7.	$\neg A$	: $\neg$INTR,4,5,6
8.	$\neg A$	: INVIATO,7

(b) Usando come premesse le prime due righe di (a), costruire una derivazione corretta di '$\neg A$'.

2. (a)

1.	$(A \vee (C \wedge D))$	: PREMESSA
2.	$(A \rightarrow (C \wedge D))$	: PREMESSA
*3.	$\neg (C \wedge D)$	: PREMESSA
*4.	$\neg A$	: MT,1,3
*5.	$(C \wedge D)$	: $\vee$ELIM,4,1
6.	$(C \wedge D)$	: RESTITUITO

(b) Usando come premesse le prime due righe di (a), costruire una derivazione corretta di '$(C \wedge D)$'.

3. (a)

1.	B	: $\wedge$ELIM,1
2.	$(B \rightarrow (A \wedge D))$	: PREMESSA
*3.	C	: ASSUNZIONE
*4.	B	: RESTITUITO,1
*5.	$(C \rightarrow B)$	: $\rightarrow$INTR,4,3
6.	$(C \rightarrow B)$	: RESTITUITO,5
**7.	$(A \wedge D)$	: $\rightarrow$ELIM,2,1
8.	$((C \rightarrow B) \wedge (A \wedge D))$	: $\vee$INTR,7,6

(b) Usando come premesse le prime due righe di (a), costruire una derivazione corretta di '$((C \rightarrow B) \wedge (A \wedge D))$'.

4. (a)

1.	$(\neg A \leftrightarrow B)$	: PREMESSA
2.	$((\neg A \rightarrow C) \wedge \neg C)$	: PREMESSA
3.	$(B \rightarrow \neg A)$	: $\leftrightarrow$ELIM,1,2
*4.	B	: INVIATO

*5. (B→¬A)	: INVIATO,3
*6. ¬A	: →ELIM,3,4
*7. (¬A→C)	: INVIATO,2
*8. C	: →INTR,6,7
*9. ¬C	: ∧ELIM,INVIATO,2
*10. ¬B	: ¬INTR,8,9,4
11. ¬B	: INVIATO,10
12. (C∧¬B)	: ∧INTR,8,9

(b) Usando come premesse le prime due righe di (a), costruire una derivazione corretta di '¬B'.

B. Aggiungere a VERIFICA-PROVA dei passi per controllare che le seguenti regole derivate siano usate correttamente:
1. modus tollens (MT)
2. sillogismo ipotetico (SI)

10.6 SUGGERIMENTI PER UNA REALIZZAZIONE SU CALCOLATORE

La conversione di VERIFICA-PROVA in un effettivo programma può risultare piuttosto difficile, perché in alcuni linguaggi richiede un pesante ricorso a funzioni di manipolazione delle stringhe, oppure, nel caso del Pascal, un uso complesso di "record". All'inizio, presumibilmente, VERIFICA-PROVA riceverà in ingresso un'intera prova, o dall'utente o da un archivio ("file"), poi verificherà la correttezza di ogni riga della prova.
Prima che questa verifica abbia luogo, occorrerà identificare i vari campi di ciascuna riga (o "record") della prova. Se si devono usare funzioni per la manipolazione delle stringhe, tale identificazione risulterà semplificata se in ogni riga si riserva un insieme ben preciso di posizioni per ciascun tipo di informazione. Ad esempio, si può imporre che le prime cinque posizioni di ogni riga siano occupate dagli asterischi (o da spazi bianchi), nell'ipotesi che non occorrano mai sottoprove a sei o più livelli di profondità. Possiamo poi collocare il numero di riga (non seguito dal punto) nelle tre posizioni successive, l'enunciato nelle posizioni dalla 9 alla 26, la regola nelle posizioni dalla 27 alla 33 e i riferimenti a righe precedenti nelle posizioni dalla 34 alla 40. Effettuando questi cambiamenti di formato, renderemo più complicata la scrittura delle prove, ma eviteremo di dover identificare i campi individuando punti, numeri e due punti tramite funzioni per la manipolazione delle stringhe.
Un altro problema può consistere nell'uso di virgole per separare i riferimenti a numeri di riga, in quanto alcuni sistemi considerano la virgola come un terminatore di stringa; in tal caso, basterà sostituire le virgole con barre (/), segni "più" (+) o addirittura spazi bianchi.
In Pascal, ogni riga può essere memorizzata in un record, avente per campi ap-

punto i campi della riga. In tal modo si semplifica VERIFICA-STRUTTURA-RIGA, ma diventa più difficile fornire in ingresso la derivazione.
Una volta ricevuta in ingresso, la prova deve essere memorizzata entro vettori ("array"), o altre strutture di dati opportune, in modo da semplificare l'individuazione delle parti di una riga. Ad esempio, se PROFONDITÀ-SOTTOPROVA(*n*) è un vettore di stringhe, si può avere:

PROFONDITÀ-SOTTOPROVA(3) = **

Ciò significa che la profondità di sottoprova della riga 3 è 2 (due asterischi). È però più opportuno memorizzare la profondità di sottoprova in un vettore numerico; con tale struttura,

PROFONDITÀ-SOTTOPROVA(3) = 2

indicherà che la profondità di sottoprova della riga 3 è 2, cioè che il numero di riga è preceduto da due asterischi.
L'enunciato e la regola di ogni riga possono essere memorizzati nei vettori di stringhe ENUNCIATO(*n*) e REGOLA(*n*), mentre i riferimenti possono essere memorizzati nei tre vettori numerici RIF1(*n*), RIF2(*n*) e RIF3(*n*).
Ad esempio, la prova

1. B	: PREMESSA
2. (B→C)	: PREMESSA
3. C	: →ELIM,2,1

verrebbe scomposta e memorizzata dal programma nel modo seguente:

PROFONDITÀ-SOTTOPROVA(1) = 0
PROFONDITÀ-SOTTOPROVA(2) = 0
PROFONDITÀ-SOTTOPROVA(3) = 0

ENUNCIATO(1) = B	REGOLA(1) = PREM
ENUNCIATO(2) = (B→C)	REGOLA(2) = PREM
ENUNCIATO(3) = C	REGOLA(3) = →ELIM

RIF1(1) = 0	RIF2(1) = 0	RIF3(1) = 0
RIF1(2) = 0	RIF2(2) = 0	RIF3(2) = 0
RIF1(3) = 2	RIF2(3) = 1	RIF3(3) = 0

Strutture di dati di questo tipo rendono molto semplici i riferimenti ai campi di ogni riga della prova. Ad esempio, poiché REGOLA(3) = →ELIM, deve essere verificata la condizione

ENUNCIATO(RIF1(3)) = (ENUNCIATO(RIF2(3))→ENUNCIATO(3))

ossia l'enunciato della prima riga citata dalla riga 3 deve essere una stringa costituita dall'enunciato della seconda riga citata dalla riga 3, seguito da una freccia e dall'enunciato della riga 3.

10.7 ESERCIZI DI PROGRAMMAZIONE

Chi abbia familiarità con la programmazione può cimentarsi negli esercizi seguenti.

A. Scegliere una struttura di dati appropriata per una riga (ad esempio, una stringa, una lista, un record o un vettore) e scrivere un algoritmo che riceva in ingresso una riga e fornisca in uscita un messaggio che dica se la riga ha un formato corretto.
B. Scrivere un programma che riceva in ingresso un'intera prova, identifichi i campi di ciascuna riga, secondo il primo formato che avevamo definito (cioè quello con punto e due punti) e memorizzi tali campi in vettori opportuni (o in strutture dati di un altro tipo).
C. Scrivere una porzione di programma che realizzi il sottoprogramma VERIFICA-STRUTTURA-RIGA, assumendo che la prova sia già stata memorizzata in una struttura di dati appropriata.
D. Scrivere una porzione di programma che realizzi il sottoprogramma VERIFICA-SOTTOPROVA, assumendo che la prova sia già stata memorizzata in una struttura di dati appropriata.
E. Assumendo che la prova sia già stata memorizzata in una struttura di dati appropriata, scrivere un programma per determinare se la regola →ELIM è stata applicata correttamente.
F. Fare lo stesso per la regola ∧INTR.
G. Fare lo stesso per la regola →INTR.
H. Assumendo, ancora una volta, che la prova sia già stata memorizzata in una struttura di dati appropriata, scrivere un programma che controlli le applicazioni delle regole ∧ELIM, ∧INTR, ¬ELIM e ¬INTR. Si tenga presente che questi connettivi sono sufficienti per esprimere tutta la logica degli enunciati, e che quindi VERIFICA-REGOLA, sotto questo aspetto, si può considerare completo, una volta che sia stata verificata la correttezza delle applicazioni di queste quattro regole.

11
Logica enunciativa: un metodo per costruire prove

Affrontiamo ora il secondo problema esposto nel Capitolo 10: come *costruire* la prova di un'argomentazione valida di cui siano assegnate le premesse e la conclusione. Sarebbe possibile definire un algoritmo per costruire una prova, nella logica degli enunciati, per qualsiasi conclusione che segua in modo valido da premesse qualsiasi. Tale algoritmo sarebbe però molto lungo, oppure "innaturale", nel senso che non seguirebbe le regole di inferenza "naturali" definite nei Capitoli 8 e 9. Nel presente capitolo, perciò, non illustreremo un algoritmo completo per la costruzione di una derivazione, bensì ci limiteremo a descrivere un metodo di costruzione di una prova che funzionerà la *maggior parte* delle volte; in certi casi, cioè, potrebbe rendersi necessario l'intervento creativo di un utente umano. Tale metodo sarà chiamato COSTRUZIONE-PROVA.

11.1 STRATEGIE GENERALI PER LA COSTRUZIONE DI PROVE

A prima vista, la costruzione della derivazione di una conclusione a partire da premesse date non assomiglia a nessuno dei problemi precedentemente affrontati. Di conseguenza, le prime volte che si deve costruire una prova ci si può sentire disorientati. In realtà la costruzione di una prova in logica assomiglia molto a problemi che risolviamo comunemente senza troppe difficoltà; essa assomiglia, inoltre, a compiti per i quali i calcolatori, e quindi gli algoritmi, vengono regolarmente impiegati.

PROBLEMI ANALOGHI

Un problema analogo alla costruzione di una prova è l'individuazione di un percorso che attraversi un labirinto.

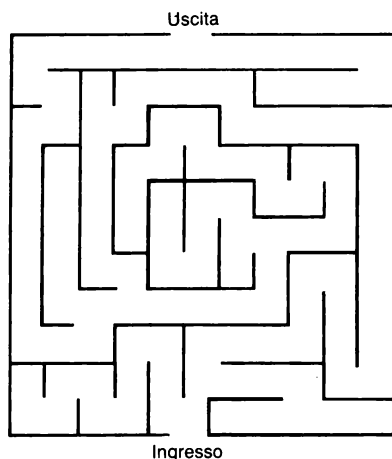



Figura 11.1 Un labirinto

È evidente che il problema consiste nel trovare un cammino che cominci all'ingresso del labirinto e termini all'uscita, senza attraversare nessuna linea (o muro). A prima vista, tale problema può non sembrare simile alla costruzione di una prova; in realtà, questi due problemi presentano analogie interessanti. In un labirinto il punto di inizio è sempre noto: è l'ingresso; anche nella costruzione di una prova si sa che si deve cominciare dalle premesse, che sono quindi il "punto di ingresso" di questo rompicapo logico. In un labirinto è altresì noto il punto che si deve raggiungere: l'uscita; anche nella costruzione di una prova lo scopo è chiaramente definito: si tratta della conclusione, che deve essere raggiunta in qualche modo. Nella ricerca di un cammino verso l'uscita di un labirinto, alcuni modi di procedere sono permessi, altri sono vietati: si può girare a destra o a sinistra, oppure si può proseguire diritto, ma non è permesso scavalcare o attraversare un muro. Nella costruzione di una prova esistono delle mosse lecite, sostituite dalle regole del sistema di deduzione naturale, cioè $\wedge$ELIM, $\rightarrow$INTR, e così via; altre mosse sono invece vietate: non è permesso inferire a piacere un enunciato qualsiasi. Come vedremo, la strategia di ricerca di un cammino verso l'uscita di un labirinto e quella di costruzione di una prova sono molto simili. Per cercare di raggiungere l'uscita di un labirinto disegnato su carta, potremmo cominciare dall'ingresso e vedere dove riusciamo ad arrivare a partire da lì; oppure potremmo considerare direttamente l'uscita, per capire come ci si può arrivare, cioè per individuare i percorsi che vi conducono. Anche per la costruzione di una prova esistono due strategie fondamentali: possiamo considerare le premesse per vedere che cosa ne consegue in base alle regole lecite, oppure possiamo prendere direttamente in considerazione la conclusione, per cercare di capire come ci si può arrivare usando le regole note.

Un altro problema analogo, di carattere più pratico, viene spesso risolto da alcuni commessi viaggiatori con l'aiuto di un calcolatore. Si supponga di dover raggiungere New Orleans in aereo partendo da Chicago, senza che esistano voli diretti nel periodo in cui si vuole fare il viaggio. Si supponga che i collegamenti aerei rilevanti seguano questo schema:

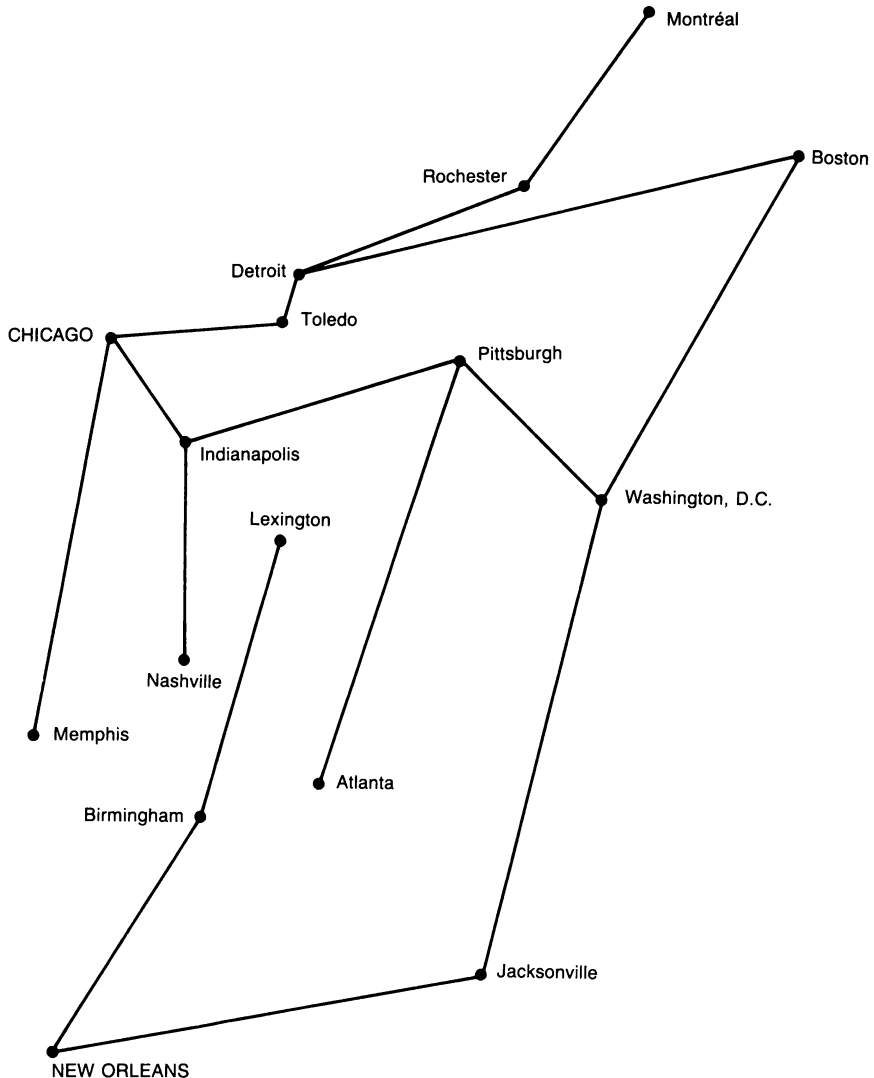



Figura 11.2 Collegamenti aerei che interessano Chicago e New Orleans

Come nel caso del labirinto e della costruzione di una prova, esistono percorsi che non portano a destinazione; viceversa, talvolta possono esistere più modi per raggiungere la destinazione: alcuni saranno più lunghi, altri più brevi. Per risolvere questo problema di collegamenti aerei possiamo usare strategie analoghe a quelle applicabili per trovare l'uscita di un labirinto: sulla mappa dei voli possiamo tracciare un percorso che parte da Chicago, oppure uno che parte da New Orleans e torna indietro verso Chicago; potremmo anche combinare le due strategie, per vedere se due percorsi cosiffatti si incontrano.
Rispetto al labirinto, il problema dei collegamenti aerei presenta alcune caratteristiche che lo rendono più simile a un problema di logica. Per raggiungere New Orleans da Chicago dobbiamo fare scalo in altre città (ad esempio, Pittsburgh) per prendere una coincidenza. Se non ci sono voli diretti per New Orleans, può rendersi necessaria anche una notevole deviazione di percorso, ad esempio attraverso Boston.
Anche nella costruzione di una prova possono mancare "voli diretti", cioè possono essere necessari percorsi verso destinazioni intermedie. Nella prova di un'argomentazione, l'equivalente di un volo diretto è una derivazione della conclusione dalle premesse effettuate mediante l'applicazione di una regola sola. Ad esempio, l'argomentazione

$(A \rightarrow B)$
A
$\therefore B$

ha una prova che corrisponde a un volo diretto:

1.	$(A \rightarrow B)$	: PREMESSA
2.	A	: PREMESSA
$\therefore$3.	B	: $\rightarrow$ELIM,1,2

In una prova, tuttavia, è raro che ci si possa concedere il lusso di un volo diretto: spesso occorre "prendere delle coincidenze", ossia dedurre enunciati intermedi che permettano di raggiungere la destinazione, cioè la conclusione.

STRATEGIE CHE PROCEDONO IN AVANTI E STRATEGIE CHE PROCEDONO ALL'INDIETRO

La strategia per costruire una prova sarà talmente simile alle strategie comunemente adottate per risolvere i problemi del labirinto e del collegamento aereo, che vale la pena di ripeterla.

> Si può cominciare dal punto di partenza e procedere in avanti, oppure si può prendere direttamente in considerazione la destinazione e procedere all'indietro.

Nella costruzione di una prova, questi due metodi corrispondono rispettivamente a:

1. Considerare le premesse e chiedersi che cosa segua da esse in base alle regole stabilite.
2. Considerare la conclusione e chiedersi come la si possa ottenere usando le regole stabilite.

Per diverse ragioni, il metodo 2, che consiste nel procedere all'indietro a partire dalla conclusione, risulterà particolarmente produttivo in logica.
Fra la costruzione di una prova e i problemi del labirinto e dei collegamenti aerei intercorre una differenza rilevante, che rende estremamente importante l'uso di una strategia per affrontare la costruzione di una prova. Tale differenza consiste nel fatto che, nel caso del labirinto e dei collegamenti aerei, esistono pochi modi per giungere dal punto di partenza alla destinazione: in un labirinto ci si può muovere in una direzione scelta a piacere, purché non si attraversi una linea; nel problema dei collegamenti aerei ci si deve spostare lungo le rotte prestabilite. Nel problema logico che consiste nel partire dalle premesse per arrivare alla conclusione, invece, ad ogni mossa si può scegliere fra più regole applicabili. Peggio ancora, nella costruzione di una prova ci sono molte più "destinazioni intermedie" che nel caso di un volo da Chicago a New Orleans; ce n'è anzi un numero infinito, e quindi non è neppure possibile rappresentare graficamente tutte le alternative, al contrario di quanto avviene nel caso dei collegamenti aerei. Inoltre, queste destinazioni intermedie sono per lo più "vicoli ciechi", che non ci fanno avvicinare alla destinazione, cioè alla conclusione. Bisogna quindi pianificare la prova con molta attenzione; per generare il piano di costruzione della prova ci occorre però una strategia, cioè, per così dire, un piano per costruire piani. COSTRUZIONE-PROVA è appunto un metodo per definire piani per costruire prove.
Si consideri l'argomentazione seguente:

A
$(A \rightarrow (B \rightarrow C))$
$\therefore (B \rightarrow C)$

Questa argomentazione è valida, ma come possiamo costruirne una prova? Cominciamo con l'esaminare le premesse, la conclusione e le loro interrelazioni. Notiamo che la conclusione è un condizionale: '$(B \rightarrow C)$'; si vede inoltre che questo stesso condizionale è una parte della seconda premessa, $(A \rightarrow (B \rightarrow C))$. Poiché il connettivo principale di questa premessa è $\rightarrow$, è chiaro che, se si potesse in qualche modo eliminare dalla premessa tale connettivo, insieme all'antecedente 'A', si rimarrebbe con il risultato desiderato, cioè $(B \rightarrow C)$. Ovviamente, per eliminare $\rightarrow$ basta usare la regola $\rightarrow$ELIM, per la cui applicazione occorre l'antecedente 'A', che infatti è fornito dalla prima premessa.

In base a queste considerazioni, possiamo produrre la prova seguente:

1. A : PREMESSA
2. (A→(B→C)) : PREMESSA
3. (B→C) : →ELIM,2,1

A questa prova si è giunti osservando che la conclusione è una sottoformula di una delle premesse e considerando come si potesse derivare questa sottoformula da sola in una riga; ma, tipicamente, una sottoformula si isola applicando una regola di tipo ELIM.
Si consideri invece l'argomentazione seguente:

B
∴((A→C)→B)

In questo caso la conclusione non è una sottoformula di una premessa: la lunga conclusione non è certo una parte della breve premessa. Che criterio possiamo adottare per determinare la strategia di costruzione della prova? Anche se non riusciamo ad individuare una relazione fra la conclusione e la premessa (al contrario del caso precedente), possiamo esaminare la struttura della conclusione in sé: si tratta ancora di un condizionale. Come si può ottenere un condizionale in una prova? Molto probabilmente, applicando →INTR; in effetti, in questo esempio è difficile concepire un modo per ottenere un condizionale senza usare →INTR. Si può dunque supporre che un passo della prova usi →INTR. Ma quali righe precedenti potrebbero permettere di ottenere '((A→C)→B)' mediante →INTR? Se si usa '(A→C)' come ASSUNZIONE e se 'B' compare in una riga successiva della stessa sottoprova, si può inferire '((A→C)→B)' con →INTR. Appare dunque logico supporre che la prova proceda nel modo seguente:

1. B : PREMESSA
*2. (A→C) : ASSUNZIONE
. .
. .
. .
*?. B : ?
*?. ((A→C)→B) : →INTR,?,?
?. ((A→C)→B) : RESTITUITO,?

I puntini e i punti interrogativi indicano parti di prova ancora da completare. L'unico problema è la derivazione di 'B' nella sottoprova, ma in questo esempio la soluzione è semplice: basta inviare la premessa nella sottoprova. La prova risultante, comprese le righe di commento, è:

1. B : PREMESSA
/INIZIO: →INTR per derivare ((A→C)→B)/

*2. (A→C) : ASSUNZIONE
*3. B : INVIATO,1
*4. ((A→C)→B) : →INTR,2,3
/FINE: →INTR per derivare ((A→C)→B)/
5. ((A→C)→B) : RESTITUITO,4

Questi due esempi permettono di delineare i fondamenti di una strategia di costruzione di una prova. Il metodo di definizione di una strategia appena applicato può essere così generalizzato:

1. Se la conclusione è una sottoformula di una riga precedente, si usa la regola appropriata di tipo ELIM per isolare tale sottoformula in una riga successiva.
2. Se la conclusione non è una sottoformula di una riga precedente, si esamina la struttura della conclusione stessa e si usa la regola appropriata di tipo INTR per ricostruire la conclusione.

Questi due criteri stanno alla base di COSTRUZIONE-PROVA.
Quando si costruisce una derivazione occorrono due elementi: la derivazione in corso, costituita dai passi di prova scritti fino all'istante corrente, e l'*agenda*, cioè la lista dei passi necessari per completare la prova. Più propriamente, l'agenda è una collezione di idee sul modo di completare la prova, ossia è poco più di un insieme di commenti.
L'agenda può essere riportata subito dopo la parte di prova già scritta, oppure può essere tenuta separata. Essa contiene delle istruzioni sulla strategia per completare la prova nel modo migliore. All'inizio, quando sono note solo le premesse e la conclusione che segue validamente da esse, la prima istruzione richiede di derivare tale conclusione; scriveremo ad esempio:

PROVA
1. A : PREMESSA
2. (A→(B→C)) : PREMESSA

AGENDA
Derivare (B→C)

L'ultima riga, 'Derivare (B→C)', è il primo obiettivo, e quindi è l'unico elemento dell'agenda all'inizio del tentativo di generare una prova.
Il resto del metodo consiste nel sostituire 'Derivare <conclusione>' con consigli più utili; corrispondentemente, le dimensioni dell'agenda aumentano. Inoltre, via via che alcuni obiettivi diventano abbastanza precisi da poter essere tradotti in righe di derivazione, aumentano anche le dimensioni della prova.

11.2 STRATEGIE DI RICERCA SU ALBERI

Uno dei principali campi d'indagine dell'Intelligenza Artificiale riguarda la *ricerca*: come programmare i calcolatori per trovare soluzioni o raggiungere obiettivi tramite metodi guidati, o "intelligenti" (compreso il metodo "a tentativo ed errore"), anziché tramite un'analisi esaustiva di tutte le possibilità. L'uso di tali metodi appare essenziale per qualunque attività che si voglia definire intelligente.
Abbiamo già evidenziato alcune analogie fra la costruzione di una prova ed altri problemi che richiedono una ricerca. Rappresentiamo ora graficamente in modo più preciso l'aspetto di una ricerca per la costruzione di una prova.
Questi diagrammi sono detti *alberi*. Il punto di partenza è costituito dall'insieme delle premesse; l'obiettivo è la conclusione. Diremo *nodo* dell'albero il punto raggiunto dopo l'applicazione corretta di una regola. In questo nodo scriviamo il nuovo enunciato derivato mediante quella regola. In ogni nodo sono disponibili tutti gli enunciati precedenti, cioè derivati "sopra" il nodo considerato, ai quali si può applicare una qualsiasi delle regole ammesse. Per ogni nodo di un albero di ricerca di una prova c'è insomma un insieme di enunciati disponibili, il quale si accresce ogni volta che viene applicata una regola; il procedimento termina quando tale insieme viene ad includere la conclusione. Questo diagramma è detto

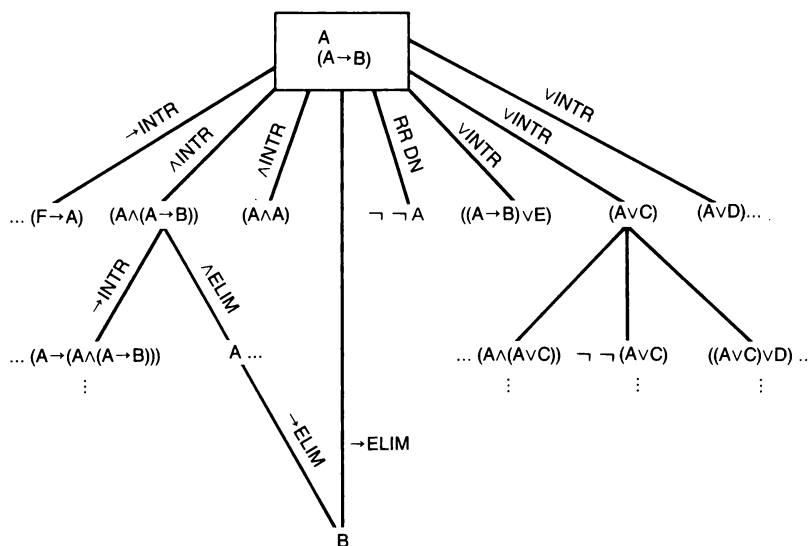



Figura 11.3 Albero di ricerca per l'argomentazione A, (A→B), ∴ B

albero di ricerca, e ciò che esso rappresenta è detto *spazio di ricerca* del problema. I possibili modi di passare dalle premesse alla conclusione sono disciplinati dalle regole disponibili. Introduciamo ora una definizione: chiameremo *distanza* di un enunciato dalle premesse (o dalla conclusione) il numero di regole usate nel cammino più breve che collega le premesse (o la conclusione) all'enunciato. Ad esempio, tutti gli enunciati sulla prima riga della figura precedente hanno distanza unitaria dalle premesse.
Possiamo ora fare alcune considerazioni sulla rappresentazione grafica, sopra riportata, di un problema di derivazione abbastanza tipico.

1. Quali che siano le premesse, esistono infiniti enunciati aventi distanza unitaria da esse; le regole ∨INTR e →INTR, infatti, permettono di aggiungere un numero indefinito di nuovi nodi.
2. Se nell'albero esiste *un* cammino che collega le premesse alla conclusione, allora ne esistono infiniti altri.
3. Se esiste un cammino di lunghezza *n* che collega le premesse alla conclusione, allora esiste anche un cammino di lunghezza maggiore.
4. Esistono infiniti enunciati aventi distanza unitaria dalla conclusione.
5. L'albero si ramifica indefinitamente in direzioni non strettamente correlate con le premesse o con la conclusione.

COSTRUZIONE-PROVA ha lo scopo di trovare un cammino ragionevolmente corto che colleghi le premesse alla conclusione. Due strategie per cercare un nodo ben preciso in un albero sono la *ricerca in ampiezza* e la *ricerca in profondità*, illustrate nella Figura 11.4.
La ricerca in ampiezza considera prima tutti i nodi aventi distanza unitaria dal nodo di partenza, poi tutti i nodi aventi distanza unitaria da questi ultimi, e così via, finché si raggiunge l'obiettivo. Si tratta di una ricerca che muove verso l'obiettivo analizzando prima tutti i nodi adiacenti a quello di partenza, poi quelli immediatamente successivi, e così via.
Si consideri l'analogia seguente. Per effettuare una ricerca in ampiezza di diecimila lire che ho perso, comincerei col visitare tutti i posti in cui potrei averle perse, e solo in caso di fallimento di queste ricerche passerei a considerare eventualità più remote (ad esempio, che mi siano cadute sul marciapiede e che il vento le abbia portate via).
Una ricerca in ampiezza nell'albero di una prova non è un metodo applicabile per generare una prova, perché, come si è visto, esistono infiniti enunciati aventi distanza unitaria dalle premesse. Dovremmo quindi considerare tutti questi enunciati, prima di continuare la ricerca, ma già questa prima fase richiederebbe un tempo infinito. Una ricerca in ampiezza, dunque, è inapplicabile quando l'albero di ricerca ha ramificazioni illimitate, permesse da regole come ∨INTR e →INTR.
Benché una ricerca in ampiezza esaustiva non sia possibile nel nostro caso, potremmo considerare una ricerca in ampiezza limitata; ad esempio, potremmo ignorare certi rami (nel gergo dell'Intelligenza Artificiale, questa tecnica è detta *potatura* dell'albero di ricerca). I rami che potremmo ignorare sono:

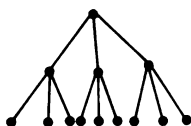
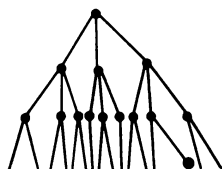
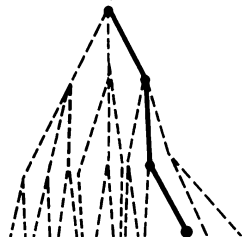



Figura 11.4 Strategie di ricerca in ampiezza e in profondità

1. Rami che usano ∨INTR o →INTR per derivare enunciati contenenti un enunciato atomico che non compare né nelle premesse né nella conclusione.
2. Rami che usano ∧INTR per derivare enunciati che non compaiono come sottoformule né nelle premesse né nella conclusione.
3. Rami che usano ∧ELIM per derivare enunciati che non compaiono come sottoformule nelle premesse o nella conclusione, né contengono una premessa o una conclusione come propria sottoformula.

Con questa ricerca in ampiezza limitata si otterrà un albero di ricerca molto "potato", che di solito permetterà di raggiungere la conclusione; spesso, tuttavia, l'albero risulta comunque molto ampio.
La seconda strategia fondamentale di ricerca è quella in profondità. Nella ricerca in profondità, si fa una sola, profonda incursione nell'albero, nella speranza di catturare la preda (la conclusione) in un colpo solo. Questa strategia si oppone alle numerose incursioni (o supposizioni), tutte ugualmente poco profonde, della ricerca in ampiezza. Fuor di metafora, la strategia di ricerca in profondità consiste nel seguire le ipotesi più promettenti, finché si riesce a formularne.

11.3 IL METODO COSTRUZIONE-PROVA

Vediamo diversi concetti che ci serviranno per definire COSTRUZIONE-PROVA. Riprendiamo innanzitutto il concetto di profondità di sottoprova, introdotto nel Capitolo 10. Ammetteremo che una sottoprova di profondità qualsiasi *contenga* se stessa, e diremo che una sottoprova S di profondità *n contiene* una sottoprova S′ di profondità maggiore di *n* se e solo se fra l'ultima riga di S e la prima riga di S′ non compare nessuna riga con meno di *n* asterischi. In particolare, la prova principale (che è una sottoprova di profondità 0) contiene se stessa e tutte le sottoprove di qualsiasi profondità.
Le sottoprove possono essere rappresentate graficamente come riquadri "annidati" (vedi Figura 11.5).
Nella Figura 11.5, la prova principale contiene se stessa e tutte e tre le sottoprove. La prima sottoprova contiene se stessa e la sottoprova 2, mentre la seconda e la terza sottoprova contengono solo se stesse.
Diremo che una riga L è *accessibile* per una successiva riga L′ se e solo se la sottoprova in cui si trova L contiene la sottoprova in cui si trova L′. Ad esempio, nella Figura 11.5 la riga 1 è accessibile per le righe di tutte le sottoprove, la riga 2 è accessibile solo per le righe delle sottoprove 1 e 2, la riga 3 è accessibile solo per le righe della sottoprova 2 e la riga 4 è accessibile solo per le righe della sottoprova 3. A volte diremo semplicemente che una riga "è accessibile". Nella terminologia informatica, un enunciato di una sottoprova si dice *locale* per tutte le sottoprove in essa contenute; perciò una riga L è accessibile per una riga L′ se e solo se l'enunciato della riga L è globale per la sottoprova in cui si trova L′, oppure L e L′ si trovano nella stessa sottoprova, nel qual caso l'enunciato è locale.

PROVA PRINCIPALE

RIGA 1: ________________

SOTTOPROVA 1

RIGA *2: ________________

SOTTO-SOTTOPROVA 2

RIGA **3: ________________

SOTTOPROVA 3

RIGA *4: ________________

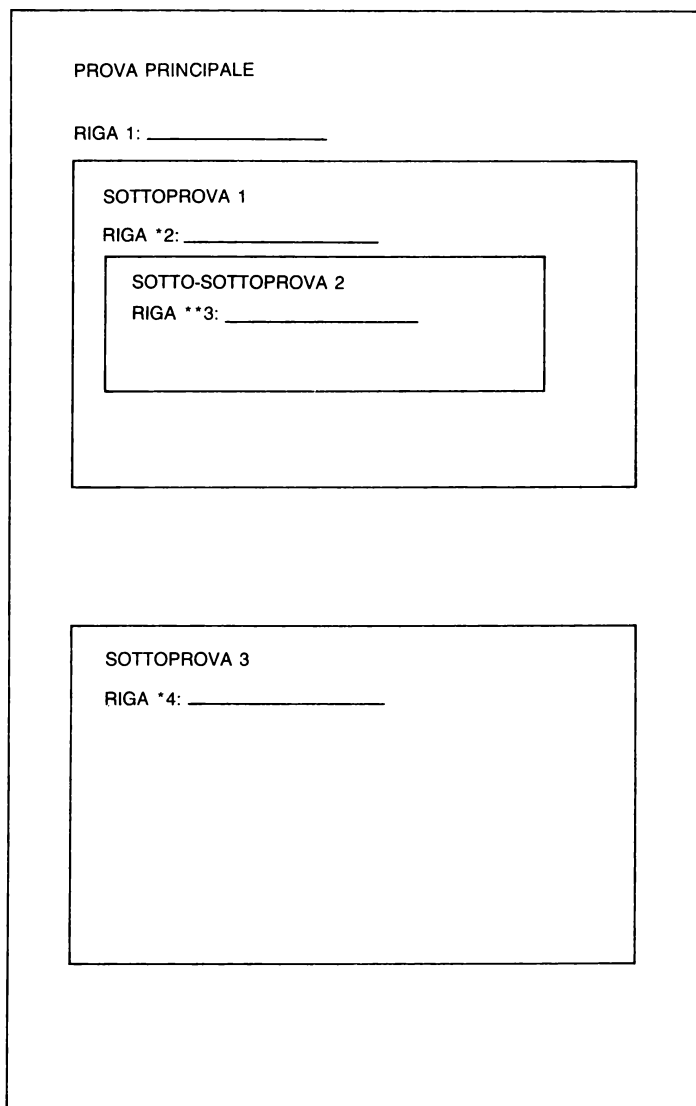



Figura 11.5 Diagramma di una prova

Nell'agenda, un asterisco prefisso a un compito indicherà che si deve cominciare una sottoprova; un enunciato fra parentesi quadre, come [**P**], indicherà che si deve citare il numero d'ordine della prima riga accessibile contenente l'enunciato **P**; chiameremo poi *obiettivo* ('**OB**') l'enunciato che compare nella prima clausola 'Derivare' dell'agenda.
Supponiamo infine di voler usare in una derivazione un enunciato **Q** che si trovi in una riga accessibile L. Se vogliamo usarlo in una riga che si trova nella stessa sottoprova di L, non dobbiamo fare niente di particolare. Se invece vogliamo usarlo in una riga di una sotto-sottoprova, dobbiamo trasferire **Q** in tale sotto-sottoprova mediante la regola INVIATO. Occorrerà inoltre effettuare alcune operazioni di aggiornamento. Per semplificare la definizione di COSTRUZIONE-PROVA, useremo un sottoprogramma chiamato 'OTTENERE **Q**':

SOTTOPROGRAMMA OTTENERE Q

1. IF la riga che contiene **Q** ha meno asterischi dell'obiettivo (cioè **Q** si trova in una sottoprova che contiene la sottoprova cui appartiene l'obiettivo)
   THEN
   (a) Sostituire la prima clausola 'Derivare' dell'agenda con INVIATO, [**Q**].
   (b) Nella sottoprova corrente, sostituire tutti i successivi riferimenti alla riga [**OB**] con il numero d'ordine di questa riga in cui compare 'INVIATO'.
   (Questa è un'operazione di aggiornamento.)
2. IF la riga che contiene **Q** ha lo stesso numero di asterischi dell'obiettivo (cioè Q si trova nella stessa sottoprova)
   THEN
   (a) Cancellare dall'agenda la prima clausola 'Derivare'.
   (b) Nella sottoprova corrente, sostituire tutti i successivi riferimenti alla riga [**OB**] con il numero d'ordine di questa riga precedente (Altra operazione di aggiornamento.)

Abbiamo ora basi sufficienti per definire il metodo.

METODO COSTRUZIONE-PROVA

1. INPUT le premesse, con la giustificazione 'PREMESSA', nel formato standard delle prove.
2. Aggiungere 'Derivare <conclusione>' all'agenda.
3. WHILE l'agenda non è vuota e contiene una clausola 'Derivare', ripetere i passi da (a) a (c):

   (a) IF **OB** = sottoformula di un enunciato **Q** di una riga accessibile
   THEN

(i) OTTENERE **Q**.
(ii) IF **OB** = **Q**.
THEN GO TO passo 3(c).
(iii) IF **OB** è una sottoformula propria di **Q**
THEN

(1) IF **Q** è un condizionale
e l'obiettivo è una sottoformula del conseguente **Q**
THEN sostituire la prima clausola 'Derivare' dell'agenda con:
Derivare <antecedente di **Q**>
Applicare →ELIM, [**Q**], [<antecedente di **Q**>]
Derivare **OB** (se **OB** ≠ <conseguente di **Q**>)
e GO TO passo 3(c).

(2) IF **Q** è una congiunzione
e l'obiettivo è una sottoformula di uno dei suoi congiunti
THEN sostituire la prima clausola 'Derivare' dell'agenda con:
Applicare ∧ELIM, [**Q**] (per ottenere il congiunto contenente **OB**)
Derivare **OB** (se **OB** ≠ congiunto inferito)
e GO TO passo 3(c).

(3) IF **Q** è una disgiunzione (**P**∨**R**)
e l'obiettivo è una sottoformula di uno dei suoi disgiunti (diciamo di **R**)
THEN sostituire la prima clausola 'Derivare' dell'agenda con:
Derivare ¬**P**
Applicare ∨ELIM, [**Q**], [¬**P**]
Derivare **OB** (se **OB** ≠ **R**)
e GO TO passo 3(c).

(4) IF **Q** è una negazione ¬**P** e **OB** è una sottoformula **R** di **P**
THEN sostituire la prima clausola 'Derivare' dell'agenda con:
*Assumere ¬**R**
*OTTENERE **Q**
*Derivare **P**
*Applicare ¬ELIM, [¬**R**], [**P**], [**Q**]
RESTITUITO, [**R**]
e GO TO passo 3(c).

(b) IF **OB** non è una sottoformula di un enunciato di una riga accessibile che soddisfi le condizioni sopra elencate
THEN

(i) IF l'obiettivo è del tipo **(P→Q)**
THEN sostituire la prima clausola 'Derivare **(P→Q)**' della agenda con:
*Assumere **P**
*Derivare **Q**
*Applicare →INTR, [**P**], [**Q**]
Applicare RESTITUITO, [(**P→Q**)]
e GO TO passo 3(c).

(ii) IF l'obiettivo è del tipo **(P∧Q)**
THEN sostituire la prima clausola 'Derivare **(P∧Q)**' della agenda con:
Derivare **P**
Derivare **Q**
Applicare ∧INTR, [**P**], [**Q**]
e GO TO passo 3(c).

(iii) IF l'obiettivo è del tipo **(P∨Q)**
THEN sostituire la prima clausola 'Derivare' dell'agenda o con:
(1) *Assumere ¬(**P∨Q**)
*Derivare **R**
*Derivare ¬**R**
*Applicare ¬ELIM, [¬(**P∨Q**)], [**R**], [¬**R**]
RESTITUITO, [(**P∨Q**)]
oppure con:
(2) Derivare **P**
Applicare ∨INTR, [**P**] per ottenere **(P∨Q)**
oppure con:
(3) Derivare **Q**
Applicare ∨INTR, [**Q**] per ottenere **(P∨Q)**
e GO TO passo 3(c).

(iv) IF l'obiettivo è del tipo ¬**P**
THEN sostituire la prima clausola 'Derivare ¬**P**' dell'agenda con:
*Assumere **P**
*Derivare **R**
*Derivare ¬**R**
*Applicare ¬INTR, [**P**], [**R**], [¬**R**]
RESTITUITO, [¬**P**]
e GO TO passo 3(c).

(v) IF non è stato eseguito nessuno dei passi precedenti
THEN sostituire la prima clausola 'Derivare **P**' dell'agenda con:
*Assumere ¬**P**
*Derivare **R**
*Derivare ¬**R**
*Applicare ¬ELIM, [¬**P**], [**R**], [¬**R**]
RESTITUITO, [**P**]
e GO TO passo 3(c).

(c) Convertire in righe della prova tutte le clausole dell'agenda che precedono la prima clausola 'Derivare' e toglierle dall'agenda.

4. STOP.

11.4 APPLICAZIONI DI COSTRUZIONE-PROVA

Applichiamo ora il metodo a un'argomentazione:

A
B
D
∴((A∧B)∧(C→D))

Eseguendo i passi 1 e 2 di COSTRUZIONE-PROVA otteniamo:

PROVA

1. A :PREMESSA
2. B :PREMESSA
3. D :PREMESSA

AGENDA

Derivare ((A∧B)∧(C→D))

L'obiettivo è '((A∧B)∧(C→D))'. Al passo 3(a) vediamo che esso non compare nelle righe precedenti (cioè, a questo punto, nelle premesse), perciò saltiamo al passo 3(b).
Poiché l'obiettivo è una congiunzione, eseguiamo il passo 3(b)(ii) e sostituiamo la clausola 'Derivare ((A∧B)∧(C→D))' dell'agenda con:

Derivare (A∧B)
Derivare (C→D)
Applicare ∧INTR, [(A∧B)], [(C→D)]

ottenendo

PROVA

1. A :PREMESSA
2. B :PREMESSA
3. D :PREMESSA

AGENDA

Derivare (A∧B)
Derivare (C→D)
Applicare ∧INTR, [(A∧B)], [(C→D)]

Andiamo ora al passo 3(c) e ritorniamo immediatamente al passo 3(a).
A questo punto l'obiettivo è '(A∧B)'. Si tratta di una congiunzione, quindi eseguiamo di nuovo il passo 3(b)(ii), ottenendo così:

PROVA

1. A :PREMESSA
2. B :PREMESSA
3. D :PREMESSA

AGENDA

Derivare A
Derivare B
Applicare ∧INTR, [A], [B]
Derivare (C→D)
Applicare ∧INTR, [(A∧B)], [(C→D)]

Dopo il passo 3(c) torniamo di nuovo al passo 3(a). Adesso però l'obiettivo è semplicemente 'A', ed è contenuto in una riga precedente; coincide infatti con la prima premessa. Il passo 3(a)(i) richiede perciò di cancellare la prima clausola 'Derivare' e di sostituire con 1 i riferimenti ad [A]. Considerando solo l'agenda, vediamo che il risultato è:

Derivare B
Applicare ∧INTR,1,[B]
Derivare (C→D)
Applicare ∧INTR, [(A∧B)], [(C→D)]

Poiché all'inizio dell'agenda c'è ancora una clausola 'Derivare', dal passo 3(c) saltiamo ancora direttamente al passo 3(a).
Adesso l'obiettivo è 'B', che coincide con la seconda premessa. Eseguendo le opportune istruzioni del passo 3(a) modifichiamo l'agenda nel modo seguente:

Applicare ∧INTR,1,2
Derivare (C→D)
Applicare ∧INTR, [(A∧B)], [(C→D)]

Ricadiamo nuovamente al passo 3(c), ma questa volta non torniamo immediatamente al passo 3(a), perché il passo 3(c) richiede di convertire in una riga della prova la prima riga dell'agenda, cioè 'Applicare ∧INTR,1,2'. Otteniamo così:

PROVA

1. A	:PREMESSA
2. B	:PREMESSA
3. D	:PREMESSA
4. (A∧B)	:∧INTR,1,2

AGENDA

Derivare (C→D)
Applicare ∧INTR,4,[(C→D)]

Ritornando al passo 3(a), vediamo che adesso l'obiettivo è '(C→D)'; poiché non è contenuto in una riga precedente (neppure nella nuova riga 4), saltiamo al passo 3(b). L'obiettivo è un condizionale; eseguendo perciò il passo 3(b)(i) modifichiamo l'agenda nel modo seguente:

*Assumere C
*Derivare D
*Applicare →INTR, [C], [D]
Applicare RESTITUITO, [(C→D)]
Applicare ∧INTR,4, [(C→D)]

Ricadendo al passo 3(c), dobbiamo convertire in una nuova riga della prova ogni clausola che nell'agenda precede la prima clausola 'Derivare'. Otteniamo così:

PROVA

1. A	:PREMESSA
2. B	:PREMESSA
3. D	:PREMESSA
4. (A∧B)	:∧INTR,1,2
*5. C	:ASSUNZIONE

AGENDA

*Derivare D
*Applicare→INTR,5,[D]

Applicare RESTITUITO, [(C→D)]
Applicare ∧INTR,4, [(C→D)]

Tornando al passo 3(a), vediamo che adesso l'obiettivo è 'D'; poiché fa parte di un enunciato precedente, eseguiamo il passo 3(a)(i), modificando l'agenda in questo modo:

*INVIATO, 3
*Applicare →INTR,5,[D]
Applicare RESTITUITO, [(C→D)]
Applicare ∧INTR,4,[(C→D)]

Ricadendo al passo 3(c), vediamo che adesso nell'agenda non ci sono più clausole 'Derivare'. Ciò significa che tutte le clausole dell'agenda devono essere convertite in righe della prova. Effettuando questa conversione dell'agenda riga per riga, otteniamo il risultato seguente:

1.	A	:PREMESSA
2.	B	:PREMESSA
3.	D	:PREMESSA
4.	(A∧B)	:∧INTR,1,2
*5.	C	:ASSUNZIONE
*6.	D	:INVIATO,3
*7.	(C→D)	:→INTR,5,6
8.	(C→D)	:RESTITUITO,7
9.	((A∧B)∧(C→D))	:∧INTR,4,8

11.5 LIMITI DI COSTRUZIONE-PROVA

COSTRUZIONE-PROVA, così come è stato definito, è quasi un vero algoritmo. Solo per alcune argomentazioni, infatti, la costruzione di una prova richiede immaginazione o ispirazione. I difetti di COSTRUZIONE-PROVA sono di tre tipi fondamentali.

In primo luogo, l'applicazione di alcune regole richiede che l'utente del metodo introduca delle supposizioni aggiuntive. È questo il caso soprattutto delle argomentazioni che conducono ai passi 3(b)(iii), 3(b)(iv) e 3(b)(v): molti passi di COSTRUZIONE-PROVA potrebbero essere trasformati direttamente in algoritmi, ma questi no. Il passo 3(b)(iii) richiede una scelta fra tre modi di introdurre una disgiunzione. I passi 3(b)(iv) e 3(b)(v) richiedono la derivazione di una contraddizione: un enunciato **R** in una riga e la sua negazione ¬**R** in un'altra; ma che enunciato può essere **R**? La risposta è lasciata alla nostra inventiva.

La seconda e la terza parte del passo 3(b)(iii) applicano la regola ∧INTR. Non esistono ambiguità riguardo a ciò che si deve derivare, ma questa tecnica non fun-

ziona sempre; quando funziona, però, funziona in modo semplice. Ci troviamo quindi di fronte a un dilemma: la prima tecnica, che richiede una prova indiretta, funziona sempre (e quindi è elencata per prima), ma richiede creatività per l'individuazione di una contraddizione; la seconda tecnica non funziona sempre, ma quando funziona non richiede creatività.
Illustriamo con due esempi i tipi di casi in cui risultano appropriate le due tecniche diverse del passo 3(b)(iii).
Si consideri l'argomentazione seguente:

$(A \wedge B)$
$\therefore$ $(A \vee C)$

Usando la prima tecnica, cominciamo con:

PROVA

1. $(A \wedge B)$:PREMESSA

AGENDA

Derivare $(A \vee C)$

In base al passo 3(b)(iii)(2) otteniamo poi:

PROVA

1. $(A \wedge B)$:PREMESSA

AGENDA

Derivare A
Applicare $\vee$INTR,[A] per ottenere $(A \vee C)$

Tornando al passo 3(a), giungiamo al passo 3(a)(iii)(2), in base al quale otteniamo:

PROVA

1. $(A \wedge B)$:PREMESSA

AGENDA

Applicare $\wedge$ELIM,1 per ottenere A
Applicare $\vee$INTR,[A] per ottenere $(A \vee C)$

Infine, dopo aver eseguito il passo 3(c), abbiamo:

PROVA

1. $(A \wedge B)$:PREMESSA

2. A :∧ELIM,1
3. (A∨C) :∨INTR,2

Si potrebbe applicare anche il passo 3(b)(iii)(1), perché funziona sempre; si potrebbe ottenere così:

PROVA

1. (A∧B) :PREMESSA
*2. ¬(A∨C) :ASSUNZIONE
*3. (A∧B) :INVIATO,1
*4. A :∧ELIM,3
*5. (A∨C) :∨INTR,4
*6. (A∨C) :¬ELIM,2,5,2
7. (A∨C) :RESTITUITO,6

In questa prova '(A∨C)' svolge il ruolo di **R**. Questa scelta di **R** provoca un doppio riferimento alla riga 2 da parte di ¬ELIM: una volta come assunzione e un'altra come parte della contraddizione. L'aspetto insolito della riga 6 di questa prova, in effetti, è un sintomo del fatto che esiste un modo più semplice per costruire una prova di '(A∨C)': la strategia fondata su ∨INTR.
A volte, però, non è possibile applicare con successo l'alternativa basata su ∨INTR del passo 3(b)(iii). In questi casi si deve ricorrere al metodo di prova più lungo e indiretto. Questa spiacevole situazione si verifica quando nessuno dei due disgiunti della disgiunzione voluta è derivabile da solo. Si consideri, ad esempio, l'argomentazione seguente:

(A∨B)
(A→C)
(B→D)
∴(C∨D)

La strategia più naturale consisterebbe nel derivare '(C∨D)' derivando prima o 'C' o 'D' e applicando poi ∨INTR. Purtroppo, però, in questo caso non si può derivare 'C' da solo, e neppure 'D'. Perciò, se cercassimo di applicare il passo 3(b)(iii)(2) a questa argomentazione, rimarremmo bloccati:

PROVA

1. (A∨B) :PREMESSA
2. (A→C) :PREMESSA
3. (B→D) :PREMESSA

AGENDA

Derivare C [si noti che 'Derivare D' andrebbe ugualmente male]
Applicare ∨INTR, [C] per ottenere (C∨D)

Come si potrebbe dimostrare mediante tavole di verità, è impossibile derivare C da queste premesse; perciò l'agenda non verrebbe mai vuotata e non si riuscirebbe mai a completare la prova a partire da essa.
Applicando il passo 3(b)(iii)(1) si riuscirebbe invece ad ottenere una prova corretta, come la seguente:

1. (A∨B) :PREMESSA
2. (A→C) :PREMESSA
3. (B→D) :PREMESSA
*4. ¬(C∨D) :ASSUNZIONE
*5. (¬C∧¬D) :RS DM,4
*6. (A→C) :INVIATO,2
*7. (B→D) :INVIATO,3
*8. (A∨B) :INVIATO,1
*9. ¬C :¬ELIM,5
*10. ¬A :MT,9,6
*11. B :∨ELIM,8,10
*12. ¬D :∧ELIM,5
*13. ¬B :MT,7,12
*14. (C∨D) :¬ELIM,4,11,13
15. (C∨D) :RESTITUITO, 14

In questa prova, la contraddizione derivata coinvolge gli enunciati 'B' e '¬B'. È però una questione di scelta (e di scoperta): sarebbe possibile derivare anche contraddizioni che coinvolgono gli enunciati 'A' e '¬A', oppure 'C' e '¬C', o anche '(A∨B)' e '¬(A∨B)'. COSTRUZIONE-PROVA ci aiuta a raggiungere la riga 4, dopodiché dobbiamo cavarcela da soli.
Neanche il passo 3(b)(iv) può essere facilmente trasformato in un procedimento automatico, perché richiede di derivare due enunciati contraddittori **R** e ¬**R**, ma non indica quali essi possano essere.
Un secondo difetto di COSTRUZIONE-PROVA consiste nel fatto che esso costruisce un'agenda verificando se un enunciato è identico a un enunciato precedente della derivazione (o a una sua sottoformula). A volte, però, un enunciato può essere non già identico a un enunciato precedente, bensì logicamente equivalente ad esso. Si consideri questa argomentazione:

(¬A∧¬B)
(¬(A∨B)→C)
∴C

Inizialmente l'agenda conterrà soltanto

Derivare C

Poiché l'enunciato 'C' è contenuto nella seconda premessa, in COSTRUZIONE-PROVA si va al passo 3(a)(iii)(1); a questo punto, l'agenda diventa:

Derivare $\neg(A \lor B)$
Applicare $\rightarrow$ELIM,2,[$\neg(A \lor B)$]

Ma come si deriva '$\neg(A \lor B)$'? In base alla lista delle equivalenze logiche, potremmo accorgerci che '$\neg(A \lor B)$' è logicamente equivalente a '$(\neg A \land \neg B)$' e può quindi essere derivato con un solo passaggio, usando la regola RS DM. Ma COSTRUZIONE-PROVA non se ne accorge, perché nota solo che i due enunciati non sono identici; perciò si va al passo 3(b)(iv) di COSTRUZIONE-PROVA. COSTRUZIONE-PROVA potrebbe essere modificato, per far sì che esso consideri le equivalenze logiche come identità e quindi usi RS. In altre parole, in ogni punto in cui COSTRUZIONE-PROVA fa riferimento ad enunciati identici lo modifichiamo in modo che faccia riferimento ad enunciati identici o logicamente equivalenti. In tal caso, però, per verificare se un enunciato è logicamente equivalente a un enunciato o a una sottoformula precedente si impiegherebbe quasi tutto il tempo necessario all'esecuzione di COSTRUZIONE-PROVA. Di conseguenza, un esecutore intelligente di COSTRUZIONE-PROVA dovrà stare all'erta per individuare i casi in cui conviene usare una regola di sostituzione, ma tali equivalenze non saranno contenute esplicitamente in COSTRUZIONE-PROVA.
Un problema ancora peggiore consiste nel fatto che a volte si può rimanere bloccati al passo 3(a), quando invece si dovrebbe andare al passo 3(b). Si consideri l'argomentazione seguente:

$(A \rightarrow (B \land C))$
B
C
$\therefore$ $(B \land C)$

Dopo aver eseguito i passi 1 e 2 di COSTRUZIONE-PROVA otteniamo:

PROVA

1. $(A \rightarrow (B \land C))$:PREMESSA
2. B :PREMESSA
3. C :PREMESSA

AGENDA

Derivare $(B \land C)$

Andiamo ora al passo 3(a), perché l'obiettivo '$(B \land C)$' è contenuto in una riga precedente (la prima premessa). Dopo l'esecuzione del passo 3(a)(iii)(1) abbiamo:

PROVA

1. $(A \rightarrow (B \land C))$:PREMESSA
2. B :PREMESSA
3. C :PREMESSA

AGENDA

Derivare A
Applicare →ELIM,1,[A]

Si può però dimostrare facilmente, ad esempio con una tavola di verità, che 'A' non segue validamente dalle premesse. È perciò impossibile derivare 'A' da solo in una riga; più precisamente, non si riuscirà mai ad eliminare tutte le clausole 'Derivare' dall'agenda di questa argomentazione, se l'agenda ha questo contenuto iniziale.
Il problema sorge al passo 3(a): quando l'obiettivo è una sottoformula di una riga accessibile, si applica il passo 3(a), mentre a volte l'obiettivo si può derivare solamente usando parti del passo 3(b). Ad esempio, se andassimo al passo 3(b)(ii), anziché al passo 3(a), otterremmo l'agenda seguente:

Derivare B
Derivare C
Applicare ∧INTR,[B],[C]

dalla quale si ricaverebbe questa prova:

1. (A→(B∧C)) :PREMESSA
2. B :PREMESSA
3. C :PREMESSA
4. (B∧C) :∧INTR,2,3

Non è facile ovviare a questo inconveniente. Praticamente, gli unici sintomi del fatto che siamo rimasti bloccati sono costituiti dalla sensazione che la prova non si stia muovendo in una direzione precisa e dal fatto che le dimensioni dell'agenda continuano ad aumentare, senza che si intraveda una fine. In questi casi si dovrebbero ripercorrere a ritroso i vari passi eseguiti, per individuare il punto in cui qualcosa nell'agenda ha cominciato ad andare storto: si tratterà di un punto in cui COSTRUZIONE-PROVA ci ha condotti al passo 3(a), quando invece sarebbe stato più proficuo trovarsi al passo 3(b)(i). Una volta individuata la probabile origine del problema, bisogna ricostruire l'agenda, andando questa volta al passo 3(b)(i), anziché al passo 3(a).
Con questa analisi del metodo COSTRUZIONE-PROVA ci siamo forse troppo immersi nei dettagli della costruzione di una prova; riemergiamone un momento, per riprendere in considerazione il significato globale dei passi di COSTRUZIONE-PROVA.
Come si era detto, alcune caratteristiche della costruzione di una prova (in particolare, il fatto che le regole ammesse permettono di collegare le premesse alla conclusione in infiniti modi) richiedono che si proceda all'indietro a partire dalla conclusione: si deve dapprima cercare di determinare il modo in cui la conclusione può essere stata ottenuta.

La conclusione si può derivare sostanzialmente in due modi: può essere una parte di una riga precedente della prova (ad esempio, di una premessa), oppure può essere ricostruita a partire da informazioni contenute nelle premesse.
I passi 3(a) e 3(b) hanno lo scopo di gestire queste possibilità. Se la conclusione è contenuta in una riga precedente, ci si ritrova al passo 3(a), che indica come estrarre la conclusione da tale riga; altrimenti ci si ritrova al passo 3(b), che contiene numerose indicazioni per costruire la conclusione a partire da informazioni contenute nelle premesse.
COSTRUZIONE-PROVA, così come è stato definito, è essenzialmente una strategia di ricerca in profondità, in quanto ci fa percorrere un unico cammino attraverso l'albero di ricerca. Il difetto principale di COSTRUZIONE-PROVA, nonché di altre strategie di ricerca in profondità, consiste nel fatto che se si è su una strada sbagliata, che non permette di raggiungere facilmente l'obiettivo, si deve tornare indietro e riconsiderare uno dei rami precedentemente ignorati. In altre parole, le strategie di ricerca in profondità richiedono spesso un ritorno all'indietro (*backtracking*).

11.6 CONCLUSIONI

In questo capitolo abbiamo definito un metodo, COSTRUZIONE-PROVA, per costruire la prova di un'argomentazione che si sa essere valida. COSTRUZIONE-PROVA non è un algoritmo, perché non funziona sempre e perché, in certi casi, può richiedere l'intervento di un essere umano; esso, tuttavia, può risultare utile, ed illustra alcune tecniche importanti. COSTRUZIONE-PROVA applica una strategia di ricerca in profondità a un albero di possibili righe di una prova, cioè segue di volta in volta l'ipotesi più promettente riguardo al modo in cui la prova deve procedere, anziché esaminare ogni volta tutte le possibilità. Questo metodo dovrebbe essere di aiuto nella costruzione di prove di argomentazioni.

11.7 ESERCIZI

A. 1. Usando COSTRUZIONE-PROVA, determinare quale dev'essere la prossima modifica della prova o dell'agenda.

a. PROVA

1. $(A \rightarrow B)$:PREMESSA
2. $(C \wedge A)$:PREMESSA

AGENDA

Derivare $(B \wedge C)$

b. PROVA

1. ((A→ ¬B)→D) :PREMESSA
2. ¬B :PREMESSA
3. (D→(E∨B)) :PREMESSA

AGENDA

Derivare D
Applicare→ELIM,3,[D]

c. PROVA

1. (A∧(B∨D)) :PREMESSA
2. ((B∨D)→(A→ ¬E)) :PREMESSA
3. A :∧ELIM,1

AGENDA

Applicare ∧ELIM,1
Applicare →ELIM,[(B∨D)],2
Applicare →ELIM,[(A→ ¬E)],3

2. In base alla prova e all'agenda di (c), determinare qual è la conclusione voluta dell'argomentazione.

B. Usando COSTRUZIONE-PROVA, costruire una prova per ciascuna delle argomentazioni seguenti:

1. (A∧(A→(B→C)))
∴(B→C)

2. ((A→B)∧(B→C))
(C→D)
∴(A→D)

3. (A∧(¬B∧C))
∴¬B

4. (A→B)
(A→ ¬E)
∴¬A

5. (A→B)
(A∧D)
∴B

6. ¬A
(A∨¬B)
(B→ ¬C)
∴(¬B∧(B→ ¬C))

7. (A→ ¬C)
(B→C)
∴(¬A∨¬B)

8. (¬B↔(A∧D))
(¬B∧(A↔E))
∴E

9. (A→(B→(¬C→D)))
(A∧B)
∴(¬C→D)

C. Aggiungere a COSTRUZIONE-PROVA un passo che tratti una clausola dell'agenda del tipo 'Derivare (**P**↔**Q**)'.

D. Per chi sa programmare:

1. Perché un "Applicare ∧INTR,*n*,*m*" che compaia nell'agenda non richiede obbligatoriamente che esista un enunciato-obiettivo?

2. Nell'ipotesi che le premesse e la conclusione contengano solo lettere proposizionali atomiche e il connettivo $\wedge$, scrivere un programma che riceva in ingresso una tale argomentazione e ne costruisca una prova.
3. Nell'ipotesi che le premesse e l'argomentazione contengano solo lettere proposizionali atomiche, parentesi e il connettivo $\rightarrow$, scrivere un programma che riceva in ingresso una tale argomentazione e ne costruisca una prova.

11.8 SUGGERIMENTI PER UNA REALIZZAZIONE SU CALCOLATORE

La realizzazione completa di COSTRUZIONE-PROVA sarà un'impresa sommamente gratificante per qualunque fanatico della programmazione. Come nel caso di VERIFICA-PROVA, occorre una struttura di dati appropriata (ad esempio, un array) per memorizzare la prova in corso. All'inizio, ovviamente, vi verranno memorizzate solo le premesse, mentre la conclusione, con lo spazio per la giustificazione lasciato vuoto, potrebbe essere temporaneamente memorizzata in una riga della prova finale avente un numero d'ordine arbitrariamente elevato (ad esempio, 100), che sia sicuramente maggiore di quelli delle altre righe della prova.
Occorre anche una struttura di dati separata per memorizzare i vari elementi dell'agenda; inoltre, poiché il contenuto dell'agenda viene continuamente modificato, bisognerà ordinare spesso tali elementi secondo la loro priorità.
Finora non abbiamo definito il formato degli elementi dell'agenda con il rigore che sarebbe necessario per permettere una realizzazione di COSTRUZIONE-PROVA su calcolatore; precisiamo dunque meglio tale formato. Le clausole 'Derivare' avranno quattro campi:

1. Un RANGO, cioè un numero che indica la priorità di un compito contenuto nell'agenda.
2. Un COMPITO che deve essere eseguito: "DERIVARE" o "APPLICARE".
3. Un enunciato da derivare (l'obiettivo).
4. La profondità di sottoprova del compito.

Si consideri, ad esempio, l'argomentazione seguente:

A
B
$\therefore$ (A$\wedge$B)

Inizialmente l'agenda avrà il seguente contenuto:

RANGO(1) = 1
COMPITO(1) = DERIVARE

ENUNC-AG(1) = (A∧B)
PROFOND-AG(1) = 0

Ciò significa che: il primo elemento dell'agenda è il primo in ordine di priorità; il compito consiste nel derivare un enunciato; l'obiettivo è la derivazione dell'enunciato '(A∧B)'; l'obiettivo non è in una sottoprova. I nomi ENUNC-AG e PROFOND-AG indicano enunciati e profondità di sottoprova *dell'agenda*.
L'altro tipo di clausola dell'agenda è 'APPLICARE', che potrà avere i campi seguenti:

1. Un RANGO.
2. Il COMPITO (in questo caso, 'APPLICARE').
3. L'obiettivo (facoltativo, eccettuato il caso delle regole ASSUNZIONE, ∧ELIM, ∨INTR).
4. La REGOLA-AG da applicare.
5. Le righe a cui si deve applicare la regola (indicate mediante numeri di riga o mediante gli enunciati che contengono).
6. La profondità di sottoprova della riga 'Applicare'.

Ad esempio,

RANGO(2)	=	3
COMPITO(2)	=	APPLICARE
ENUNC-AG(2)	=	(A∨B)
REGOLA-AG(2)	=	∨INTR
RIF1-AG(2)	=	1
RIF2-AG(2)	=	0
RIF3-AG(2)	=	0
PROFOND-AG(2)	=	1

significherà che: la priorità della riga 2 dell'agenda è 3; il compito è APPLICARE; l'obiettivo è '(A∨B)'; la regola deve essere applicata alla riga 1; APPLICARE è in una sottoprova di profondità 1.
Si noti che il compito può essere memorizzato in forma numerica, perché i compiti possibili sono solo due; ad esempio, 1 = DERIVARE, 2 = APPLICARE. Analogamente, anche REGOLA-AG può essere memorizzata in forma numerica: 1 per ∧INTR, 2 per ∧ELIM, 3 per ∨ELIM, eccetera. Usando numeri dove è possibile, si evitano alcune fastidiose operazioni su stringhe. Non si può invece convertire facilmente in forma numerica un enunciato, come, ad esempio, '(A∧(B∨C))'.
Dopo che l'intelaiatura è stata così definita, la conversione del metodo COSTRUZIONE-PROVA in un programma per calcolatore è relativamente semplice; alcune caratteristiche del problema, tuttavia, potrebbero ostacolarla.

1. A volte una riga dell'agenda deve essere sostituita da due o più righe. Queste nuove righe vengono sempre inserite all'inizio; perciò alterano l'ordine degli elementi dell'agenda. La loro priorità nell'agenda è indicata dal loro

RANGO. Se un compito di rango 1 deve essere sostituito da due compiti, a questi si possono assegnare i ranghi 1.1 e 1.2. Dopo aver cancellato il compito originario, sappiamo che il compito di rango 1.1 deve essere eseguito prima del compito di rango 1.2 e che entrambi devono essere eseguiti prima di qualsiasi compito di rango 2.
Analogamente, un compito di rango 2.1 può essere sostituito da compiti di rango 2.1.1, 2.1.2 e 2.1.3. Indicando esplicitamente il RANGO di un compito si effettua un ordinamento automatico degli elementi dell'agenda.

2. Come si è visto, a volte COSTRUZIONE-PROVA prende una strada sbagliata, nel qual caso l'agenda cresce in modo spropositato e non si riesce ad avvicinarsi alla conclusione. Poiché i calcolatori operano molto velocemente, in alcuni microsecondi l'agenda può ritrovarsi piena di centinaia di compiti che non permetteranno mai a COSTRUZIONE-PROVA di raggiungere la conclusione.
   Per questo problema esistono due soluzioni: una piuttosto brutale, l'altra più elegante.
   Potremmo scrivere il programma in modo che, quando le dimensioni della lista superano un certo limite (ad esempio, venti o trenta righe), esso si arresti e avverta l'utente dell'insorgere del problema; si tratta del cosiddetto "blocco per incidente".
   Un'altra soluzione consiste nel comunicare all'utente ogni modifica dell'agenda e chiedergli se vuole:
   a. Continuare.
   b. Fermarsi.
   c. Risalire a uno stadio precedente della prova e dell'agenda.
   d. Saltare il passo 3(a).
   e. Fornire un proprio suggerimento su come continuare: che contraddizione si deve cercare di derivare, che passo si deve eseguire, oppure che obiettivo si deve fissare.
3. Poiché può essere necessario risalire a uno stadio precedente, occorre memorizzare, oltre al contenuto corrente della prova e dell'agenda, anche i contenuti passati: se si vuole essere in grado di risalire a uno stadio passato, bisogna avere a disposizione la prova e l'agenda di ogni passaggio precedente. Il modo più semplice per ottenere questo risultato consiste nel memorizzare le prove e le agende entro strutture di dati adatte, come matrici bidimensionali. Ad esempio, REGOLA-AG(3,2) sarà la REGOLA-AG del secondo elemento della terza agenda generata durante l'esecuzione di COSTRUZIONE-PROVA.
4. A COSTRUZIONE-PROVA si possono aggiungere altri criteri e altre strategie, oltre a quelle indicate nel presente capitolo. Ad esempio, si potrebbero indicare strategie per scegliere le contraddizioni da derivare per ¬ELIM e ¬INTR, nonché accorgimenti per ottenere prove più brevi. Inoltre, se le argomentazione di cui COSTRUZIONE-PROVA deve fornire una prova sono tutte create da una stessa persona, si possono anche introdurre strategie per cercare di identificare le abitudini di questa persona; essa potrebbe, ad esempio, usare ∨INTR spesso, o →INTR di rado.

12
Logica predicativa: quantificazione

I procedimenti e le tecniche finora studiate per dimostrare la validità delle argomentazioni funzionano correttamente solo in un ambito limitato, perché esistono molte argomentazioni valide la cui validità non può essere dimostrata con i metodi che abbiamo studiato nei capitoli precedenti. Si consideri, come esempio, la seguente argomentazione, spesso riportata dai testi di logica:

Tutti i Greci sono mortali.
Socrate è greco.
∴ Socrate è mortale.

Poiché questa argomentazione contiene solo enunciati atomici, tutti diversi fra loro, essa verrebbe rappresentata simbolicamente mediante tre lettere proposizionali distinte:

G
C
∴ A

Questa argomentazione ha la struttura seguente:

P
Q
∴ **R**

In logica enunciativa, però, non si può dimostrare la validità di un'argomentazione di questo tipo, se **R** è un enunciato atomico diverso da **P** e **Q**. Tuttavia, questa argomentazione sembra proprio valida: appare impossibile che la conclusione sia falsa quando le premesse sono vere.

12.1 INDIVIDUI E PROPRIETÀ

Se consideriamo più attentamente l'argomentazione precedente, vediamo che la sua validità dipende dal fatto che tutti gli individui di un certo tipo hanno una particolare proprietà, e che uno specifico individuo, Socrate, è un individuo di quel tipo; perciò, conclude l'argomentazione, questo specifico individuo (Socrate) ha quella particolare proprietà. In base a questa considerazione, appare opportuno estendere il nostro linguaggio simbolico, includendovi dei meccanismi che permettano di far riferimento a degli individui e di indicare le proprietà possedute dagli individui. Nella logica enunciativa consideravamo gli enunciati atomici o molecolari solo nel loro complesso; adesso, nella logica predicativa, considereremo ciò che alcuni enunciati precisano a proposito di determinati individui.

COSTANTI INDIVIDUALI

Un *individuo* è una singola entità specifica, come una persona, un numero, un cavallo, e così via. Per poter far riferimento a degli individui, modificheremo le convenzioni sull'uso delle lettere dell'alfabeto. D'ora in poi useremo lettere minuscole corsive dell'inizio dell'alfabeto

$a, b, c, \ldots, h$

per indicare (o "far riferimento a") degli individui. Queste lettere saranno chiamate *costanti individuali*; sono dette costanti perché entro qualsiasi argomentazione fanno riferimento sempre allo stesso individuo. Se serve un numero maggiore di costanti individuali, aggiungeremo dei numeri a destra delle lettere; ad esempio, anche '$a1$', '$b12$' e '$c567$' sono costanti individuali. Nell'argomentazione che conclude che Socrate è mortale, possiamo usare 'c' per indicare 'Socrate'.
Gli individui hanno delle *proprietà*, come l'essere mortale, o divisibile per 2, o rosso, o tappezzato, eccetera. D'ora in poi useremo le lettere maiuscole dell'inizio dell'alfabeto

A, B, ..., O

per indicare proprietà di individui, oltre che per indicare enunciati atomici semplici. Ad esempio, potremmo usare 'G' per indicare la proprietà di essere greco. Tali indicatori di proprietà sono detti *lettere predicative*. Se occorre un numero maggiore di lettere predicative o di enunciati atomici, aggiungeremo, come sempre, un numero a destra della lettera.
Il fatto che uno specifico individuo possiede una particolare proprietà sarà rappresentato scrivendo la costante che indica l'individuo subito dopo la lettera predicativa che indica la proprietà. 'Socrate è greco' si scriverà perciò:

Gc

Se 'A' è la lettera predicativa che indica la proprietà di essere saggio, allora 'Socrate è saggio' si scriverà

A*c*

Se '*a*' è la costante individuale che indica Aristotele, allora

(G*a*∧A*a*)

significa che Aristotele è greco e saggio.

VARIABILI

Spesso, tuttavia, si fanno affermazioni che non riguardano nessun individuo particolare. Ad esempio, la prima premessa dell'argomentazione iniziale riguarda tutti i Greci; per fare un altro esempio, se diciamo

1. Qualcuno ha preso 10 nel compito in classe.

non ci riferiamo esplicitamente a nessun individuo specifico. Può darsi che non sappiamo esattamente di quale individuo si tratti, oppure che lo sappiamo ma non vogliamo dirlo. Per esprimere queste affermazioni nel nostro linguaggio simbolico, dobbiamo modificare il modo di formare enunciati ora visto, per poter costruire enunciati "incompleti", che useremo poi per generare nuovi enunciati. Si consideri l'enunciato seguente:

Enrico ha preso 10 nel compito in classe.

Se appena sarà possibile, useremo lettere che ricordino il corrispondente italiano degli individui e delle proprietà. In questo caso, ad esempio, scegliendo in modo ovvio la costante individuale e la lettera predicativa, scriveremo l'enunciato

2. D*e*

Per rappresentare l'enunciato (1), dobbiamo innanzitutto eliminare il preciso riferimento ad Enrico nell'enunciato (2). Come si è detto, può darsi che non vogliamo identificare la persona che ha preso 10, ma che vogliamo ugualmente dire che qualcuno l'ha preso. Se però ci limitassimo a cancellare la '*e*', non si capirebbe più che in quella posizione deve comparire un simbolo; stabiliremo perciò che alcuni simboli possano essere usati come "segnaposto" negli enunciati. Questi segnaposto, detti *variabili individuali*, saranno rappresentati da lettere minuscole corsive della fine dell'alfabeto:

u, *v*, *w*, *x*, *y*, *z*

Si noti che le variabili individuali non indicano nessun individuo specifico: servono solo come segnaposto, cioè come riferimento a un individuo non specificato. Come nel caso delle costanti individuali, se occorre un numero maggiore di variabili individuali aggiungeremo un numero a destra della lettera: *x*3, *y*8, *z*52.
Il primo passo per generare un enunciato che non nomina un individuo specifico consiste nel costruire un *enunciato incompleto*, ottenuto sostituendo le costanti individuali con variabili individuali. Dunque, per generare l'enunciato (1) a partire dall'enunciato (2), cominciamo col sostituire la costante individuale '*e*' con, ad esempio, la variabile individuale '*x*'. Otteniamo così:

D*x*

che si può leggere

x ha preso 10 nel compito in classe.

Questo enunciato è però incompleto, e il suo valore di verità non è né TRUE né FALSE. Chiameremo *formule* questi enunciati incompleti (questa definizione di "formula" è diversa da quella introdotta nel Capitolo 5).

12.2 QUANTIFICATORI

Chiameremo *quantificatore* un'espressione che indica quanti individui, ma non quali, hanno la proprietà indicata da un enunciato incompleto. Un quantificatore premesso a un enunciato incompleto lo rende completo, cioè lo trasforma in un enunciato il cui valore di verità è o TRUE o FALSE. Benché esistano diversi quantificatori, noi ne useremo solo due: il *quantificatore universale* si usa per esprimere il fatto che tutti gli individui hanno la proprietà indicata, mentre il *quantificatore esistenziale* si usa per esprimere il fatto che qualche individuo (almeno uno) ha tale proprietà.
Come si rappresentano i quantificatori? Il quantificatore universale è costituito da una A rovesciata posta a sinistra di una variabile: $\forall x$, $\forall w$, $\forall z$. Il quantificatore esistenziale è invece rappresentato da una E ribaltata posta a sinistra di una variabile: $\exists x$, $\exists y$, $\exists w$. L'enunciato voluto si costruisce ponendo il quantificatore appropriato a sinistra della formula opportuna. Un quantificatore è appropriato se contiene la stessa variabile che compare nella formula. Perciò l'enunciato (1), 'Qualcuno ha preso 10 nel compito in classe', si rappresenta così:

$\exists x$D*x*

mentre

Tutti hanno preso 10 nel compito in classe.

si rappresenta così:

$\forall x Dx$

Queste due rappresentazioni simboliche di solito si leggono, rispettivamente, nel modo seguente:

Esiste almeno un individuo *x* tale che *x* ha preso 10 nel compito in classe.

e

Per ogni individuo *x*, *x* ha preso 10 nel compito in classe.

FORMULE BEN FORMATE

Dobbiamo precisare in modo rigoroso le forme degli enunciati; a tal fine definiremo il concetto di *formula ben formata*, o *fbf*. Come si ricorderà, una stringa è una sequenza qualsiasi di caratteri. Nel Capitolo 5 avevamo detto che una formula è una stringa contenente numeri, parentesi e simboli di connettivi, disposti in un modo ben preciso.
Definiamo ora che cos'è una formula ben formata:

1. Una stringa costituita da una sola lettera proposizionale è una fbf.
2. Una stringa costituita da una lettera predicativa seguita da una costante o da una variabile individuale è una fbf.
3. Se **P** e **Q** sono fbf, allora lo sono anche

   $\neg \mathbf{P}$
   $(\mathbf{P} \wedge \mathbf{Q})$
   $(\mathbf{P} \vee \mathbf{Q})$
   $(\mathbf{P} \rightarrow \mathbf{Q})$
   $(\mathbf{P} \leftrightarrow \mathbf{Q})$
   (La lista dei connettivi ammessi può essere estesa.)
4. Se **P** è una fbf, allora $\forall \mathbf{v} \mathbf{P}$ e $\exists \mathbf{v} \mathbf{P}$ (dove **v** è una variabile qualsiasi: *x*, *y*, *z*, *w*, ...) sono fbf.

Ad esempio, le stringhe seguenti sono tutte formule ben formate:

$\forall x Fx$
$(\forall x Fx \wedge Ga)$
$\forall x(Fx \wedge Ga)$
$\exists x \neg (Gx \vee B)$
$(\forall y Hy \rightarrow \exists y Gy)$
$(A \rightarrow \neg B)$
$(Fx \vee Gy)$
$\forall x(Fx \rightarrow Hz)$
$\forall x(Fx \rightarrow A)$
$(\exists x Gx \wedge Hx)$

Più avanti amplieremo il concetto di formula ben formata per comprendervi le relazioni, cioè le lettere predicative seguite da una stringa contenente più di una costante o variabile individuale.

CAMPO DI UN QUANTIFICATORE

Per trattare gli enunciati quantificati e le formule dobbiamo introdurre qualche altra definizione. Ogni quantificatore ha un *campo di applicazione*, spesso detto semplicemente *campo*. Intuitivamente si può dire che il campo è la formula coperta dal quantificatore; più precisamente, chiameremo campo di un quantificatore la formula ben formata immediatamente a destra del quantificatore stesso. Ad esempio

Il campo di	nell'enunciato	è la formula
$\forall x$	$\forall x(\mathrm{A}x \rightarrow \mathrm{B}x)$	$(\mathrm{A}x \rightarrow \mathrm{B}x)$
$\exists x$	$(\exists x \mathrm{A}x \wedge \mathrm{C})$	$\mathrm{A}x$
$\forall y$	$\exists x \forall y(\mathrm{A}x \wedge (\mathrm{B}y \rightarrow \mathrm{C}y))$	$(\mathrm{A}x \wedge (\mathrm{B}y \rightarrow \mathrm{C}y))$
$\exists x$	$\exists x \forall y(\mathrm{A}x \wedge (\mathrm{B}y \rightarrow \mathrm{C}y))$	$\forall y(\mathrm{A}x \wedge (\mathrm{B}y \rightarrow \mathrm{C}y))$
$\forall z$	$\forall z \neg(\mathrm{A}z \vee \mathrm{B}z)$	$\neg(\mathrm{A}z \vee \mathrm{B}z)$

OCCORRENZE LIBERE E LEGATE DI VARIABILI

Negli enunciati sopra riportati, la stessa variabile compare in più posti diversi. Ogni volta che una variabile compare in una formula si ha un'*occorrenza* distinta di tale variabile nella formula. Un'occorrenza di una variabile si dice *legata* se compare nel campo di un quantificatore contenente tale variabile; altrimenti l'occorrenza si dice *libera*. In una data formula, una stessa variabile può avere sia occorrenze libere, sia occorrenze legate.

Nella formula	la variabile	compare
$\forall x \forall y(\mathrm{F}y \rightarrow \mathrm{G}x)$	y	legata
$\forall x(\mathrm{F}x \rightarrow \mathrm{G}y)$	y	libera
$(\mathrm{F}x \wedge \forall x \mathrm{G}x)$	x	prima libera, poi legata

A questo punto è opportuno fare due considerazioni. In primo luogo, le regole che definiscono le fbf permettono la cosiddetta quantificazione inutile: a una fbf si può prefiggere anche un quantificatore contenente una variabile che non compare, o non compare libera, nella fbf. Vediamo alcuni esempi di fbf quantificate inutilmente:

$\forall x \mathrm{F} y$
$\exists y \mathrm{A}$
$\forall z(\exists z \mathrm{H} z \wedge \mathrm{G} x)$

In secondo luogo, un *enunciato* è una fbf priva di occorrenze libere di variabili. Le formule ben formulate che contengono occorrenze libere di variabili non hanno un valore di verità: sono enunciati incompleti. Invece le fbf prive di occorrenze libere di variabili sono enunciati e hanno un valore di verità definito.

UN ALGORITMO PER GLI ENUNCIATI

L'algoritmo del Capitolo 5 che determina se una stringa è un enunciato ben formato deve essere modificato, perché abbiamo introdotto delle possibilità nuove. In quell'algoritmo, dopo aver sostituito ogni enunciato atomico con un valore di verità, e dopo aver cambiato ogni '¬0' in '1' e ogni '¬1' in '0', dovevamo avere formule tutte del tipo

<valore di verità>

oppure

(<valore di verità 1> ☆ <valore di verità 2>)

Adesso abbiamo due possibilità nuove: un enunciato può cominciare con uno dei due quantificatori. Chiamiamo *formula quantificata* una fbf i cui primi due simboli sono, nell'ordine, un quantificatore e una variabile. Nell'algoritmo del Capitolo 5, dopo aver sostituito gli enunciati atomici con i loro valori di verità, potremo avere dunque:

<formula quantificata>
(<formula quantificata> ☆ <valore di verità>)
(<valore di verità> ☆ <formula quantificata>

oppure

(<formula quantificata 1> ☆ <formula quantificata 2>)

Queste strane combinazioni, però, si possono evitare se, prima di sostituire gli enunciati atomici con valori di verità, si effettuano le operazioni seguenti:

Passo 0. FOR ogni formula quantificata della stringa, da sinistra a destra
a. Cancellare i primi due simboli.

b. Sostituire tutte le occorrenze ora libere della variabile cancellata con una costante individuale non usata precedentemente.
c. Sostituire ogni formula predicativa contenente solo costanti individuali con una lettera proposizionale atomica non usata precedentemente.

Ad esempio, effettuando queste operazioni sull'enunciato

$(A \land \forall x Fx)$

si ottiene:

$(A \land Fx)$
$(A \land Fa)$
$(A \land B)$

A questo punto si può proseguire con CALCOLO DEL VALORE DI VERITÀ, sostituendo con dei valori di verità i nuovi enunciati atomici così introdotti. Se in un qualunque punto di CALCOLO DEL VALORE DI VERITÀ non si ha un singolo valore di verità o una coppia di valori di verità uniti da un connettivo, ciò significa che la stringa non era un enunciato. Si noti, tuttavia, che il valore di verità ottenuto con il precedente algoritmo è privo di significato, e quindi non può essere usato in DETERMINAZIONE DI VALIDITÀ/NON VALIDITÀ. Dobbiamo raffinare un passo del nostro nuovo algoritmo: come possiamo identificare in modo automatico le "occorrenze ora libere", che erano legate prima della cancellazione del quantificatore? A tal fine dobbiamo identificare la fbf che segue il quantificatore eliminato. Il simbolo che segue la variabile del quantificatore può essere soltanto (1) una lettera predicativa, (2) un segno di negazione, (3) una parentesi aperta, oppure (4) un quantificatore (escludiamo la quantificazione inutile di una lettera proposizionale). Ciascuno di questi casi si può trattare facilmente. Si noti che quando il campo comprende una formula con parentesi, occorre il "primo contatore" per trovare la parentesi chiusa corrispondente. Consideriamo un esempio più complicato:

$\forall x(\exists y(Fx \lor Ly) \rightarrow (Gx \land \forall y Hy))$
$(\exists y(Fx \lor Ly) \rightarrow (Gx \land \forall y Hy))$
$(\exists y(Fa \lor Ly) \rightarrow (Ga \land \forall y Hy))$
$(\exists y(A \lor Ly) \rightarrow (B \land \forall y Hy))$
$((A \lor Lb) \rightarrow (B \land \forall y Hy))$
$((A \lor C) \rightarrow (B \land \forall y Hy))$
$((A \lor C) \rightarrow (B \land Hc))$
$((A \lor C) \rightarrow (B \land D))$

12.3 VALORI DI VERITÀ DI ENUNCIATI QUANTIFICATI

Prima di descrivere le regole di inferenza per l'introduzione e l'eliminazione dei quantificatori nel corso di una derivazione, dobbiamo definire in modo rigoroso il concetto di valore di verità per gli enunciati contenenti dei quantificatori. Gli enunciati come 'Aristotele è greco', cioè 'G*a*', non costituiscono un problema: 'G*a*' vale TRUE se l'individuo rappresentato da '*a*', cioè Aristotele, ha la proprietà indicata da 'G', cioè l'essere greco; considerazioni perfettamente analoghe valgono per tutti gli enunciati contenenti solamente costanti individuali e lettere predicative. 'G*a*' vale FALSE in qualsiasi altra situazione, cioè quando l'individuo rappresentato da '*a*' non possiede la proprietà indicata da 'G'. Un *esemplare* di una fbf è un enunciato ottenuto sostituendo tutte le occorrenze libere di variabili individuali con costanti individuali.

Usando la funzione di valutazione V precedentemente introdotta, affermiamo che:

V(∀*x*G*x*) = TRUE se e solo se tutti gli esemplari di 'G*x*' hanno valore di verità TRUE.
V(∃*x*G*x*) = TRUE se e solo se almeno un esemplare di 'G*x*' ha valore di verità TRUE.

Si ricordi che se il valore di verità di un enunciato non è TRUE, è FALSE. L'insieme di individui di cui si sta parlando è detto *universo del discorso*. Ad esempio, se l'universo del discorso contiene gli individui rappresentati da '*a*', '*b*' e '*c*', allora 'F*a*', 'F*b*' e 'F*c*' sono tutti gli esemplari di 'F*x*'; perciò V(∀*x*F*x*) = TRUE se e solo se V(F*a*) e V(F*b*) e V(F*c*) sono TRUE, mentre V(∃*x*F*x*) = TRUE se e solo se V(F*a*) o V(F*b*) o V(F*c*) è TRUE.

12.4 QUANTIFICAZIONE DI FORMULE MOLECOLARI

Riprendiamo l'esempio che affermava che Aristotele è greco e saggio: si tratta della congiunzione dei due enunciati 'Aristotele è greco' e 'Aristotele è saggio'. Useremo perciò il simbolo della configurazione, scrivendo:

(G*a*∧A*a*)

A partire da questo enunciato possiamo costruire una fbf sostituendo tutte le occorrenze della costante individuale '*a*' con la variabile '*x*'; otteniamo così:

(G*x*∧A*x*)

QUANTIFICATORI ESISTENZIALI

Se facciano precedere l'espressione precedente da un quantificatore esistenziale riguardante 'x', otteniamo un nuovo enunciato:

$\exists x(\mathrm{G}x \wedge \mathrm{A}x)$

che si può leggere

Esiste qualcosa che è Greco e saggio.

oppure, in modo più naturale,

Qualche Greco è saggio.

Abbiamo cosí un modello generale per rappresentare gli enunciati del tipo:

Qualche **S** è **T**.
Alcuni **S** sono **T**.

La rappresentazione simbolica sarà:

$\exists x(\mathbf{S}x \wedge \mathbf{T}x)$

che, alla lettera, si legge "Esiste almeno un x tale che x è **S** e x è **T**".
Questo modello è dunque adatto per tutti gli enunciati seguenti:

Qualche bambino è grazioso.
Alcuni bambini sono graziosi.
Alcuni studenti sono musicisti.
Alcuni musicisti sono studenti.
Alcuni studenti hanno suonato tutta notte.
(cioè alcuni studenti sono persone che suonano tutta notte)

Analogamente, gli enunciati del tipo

Qualche **S** non è **T**.
Alcuni **S** non sono **T**.

si rappresenteranno simbolicamente così:

$\exists x(\mathbf{S}x \wedge \neg \mathbf{T}x)$

che, letteralmente, si legge "Esiste almeno un x tale che x è **S** e x non è **T**".

QUANTIFICATORI UNIVERSALI

Quantificando universalmente in modo appropriato la formula ben formata

$(Gx \wedge Ax)$

si ottiene

$\forall x(Gx \wedge Ax)$

che si può leggere.

Tutto è sia Greco che saggio.

Raramente faremo un'affermazione del genere. Piuttosto, ci interesserà esprimere il fatto che tutti i Greci sono saggi. Dobbiamo quindi rappresentare simbolicamente un enunciato del tipo

Tutti gli S sono **T**.

In questo caso affermiamo che la proprietà **T** vale per tutti gli individui di tipo S. Vedremo ora che questo tipo di affermazione può essere rappresentato da un condizionale; in forma simbolica, cioè,

Tutti gli **S** sono **T**.

si scriverà

$\forall x(Sx \rightarrow Tx)$

Ciò significa che per ogni *x*, se *x* è **S**, allora *x* è **T**. In base alla regola di valutazione, questo enunciato vale TRUE se ogni esemplare di '$(Sx \rightarrow Tx)$' vale TRUE. Alcuni di questi esemplari possono essere:

$(Sa \rightarrow Ta)$
$(Sb \rightarrow Tb)$
$(Sc \rightarrow Tc)$
$(Sd \rightarrow Td)$

Questi esemplari sono degli enunciati, ma che cosa affermano? Il primo afferma che se *a* ha la proprietà **S**, allora *a* ha la proprietà **T** (ad esempio, se è greco, allora è saggio). Quando vale TRUE questo enunciato? Ovviamente, se 'Sa' e 'Ta' valgono entrambi TRUE, allora l'enunciato condizionale '$(Sa \rightarrow Ta)$' vale TRUE. Allo stesso modo, la nota tavola di verità del condizionale ci dice quando vale TRUE e quando vale FALSE ognuno degli esemplari. Come è noto, il condizionale vale

FALSE esattamente quando l'antecedente vale TRUE e il conseguente vale FALSE. Con riferimento al solito esempio, un esemplare vale FALSE solo quando almeno un individuo rappresentato da una costante individuale è Greco ma non è saggio; ma un Greco che non è saggio è proprio la situazione che falsifica l'enunciato originario 'Tutti i Greci sono saggi'.
È importante capire come funziona il condizionale nel caso di enunciati quantificati universalmente. Il valore di verità di

Tutti gli hamburger sono deliziosi.

è TRUE se e solo se ogni e qualsiasi hamburger è delizioso. Perciò, quando esaminiamo l'universo del discorso un individuo per volta, dobbiamo accertarci che, se un individuo è un hamburger, allora è delizioso. Se l'individuo considerato non è un hamburger, allora l'esemplare non vale FALSE, cioè vale TRUE; si noti che in un esemplare di quest'ultimo tipo l'antecedente vale FALSE. Solo se troviamo un individuo che è un hamburger ma non è delizioso l'esemplare vale FALSE. Se anche un solo esemplare vale FALSE, l'enunciato quantificato universalmente vale FALSE. Ad esempio:

$\forall x(\mathrm{H}x \rightarrow \mathrm{D}x)$

Universo del discorso:

	indicato con
un hamburger delizioso	*a*
una mela deliziosa	*b*
una mela marcia	*c*
Giulio Cesare	*d*

Esemplari:

$(\mathrm{H}a \rightarrow \mathrm{D}a)$	TRUE
$(\mathrm{H}b \rightarrow \mathrm{D}b)$	TRUE
$(\mathrm{H}c \rightarrow \mathrm{D}c)$	TRUE
$(\mathrm{H}d \rightarrow \mathrm{D}d)$	TRUE

Perciò in questo universo del discorso '$\forall x(\mathrm{H}x \rightarrow \mathrm{D}x)$' vale TRUE. Consideriamo invece il seguente universo del discorso:

	indicato con
un hamburger delizioso	*a*
una mela deliziosa	*b*
un hamburger andato a male	*c*
Giulio Cesare	*d*

Esemplari:

$(Ha \rightarrow Da)$	TRUE
$(Hb \rightarrow Db)$	TRUE
$(Hc \rightarrow Dc)$	FALSE
$(Hd \rightarrow Dd)$	TRUE

In questo caso, l'enunciato quantificato universalmente vale FALSE, perché un esemplare vale FALSE.
Abbiamo ora una modalità generale di rappresentazione di enunciati del tipo

Tutti gli **S** sono **T**.

Scriveremo dunque

$\forall x(\mathbf{S}x \rightarrow \mathbf{T}x)$

per rappresentare enunciati come

Tutti i guidatori sono prudenti.
Tutti gli atleti sono sani.
Tutti i pesci nuotano.
(cioè tutti i pesci sono nuotatori)

Un enunciato del tipo

Nessun **S** è **T**.

significa che ogni singolo individuo che è **S** non è **T**, e può essere riscritto nella seguente forma:

Tutti gli **S** non sono **T**.

Scriveremo dunque

$\forall x(\mathbf{S}x \rightarrow \neg \mathbf{T}x)$

per rappresentare enunciati come

Nessun cobra è un animale domestico.
Nessun Australiano è triste.
Nessun vegetariano mangia carne.
(cioè nessun vegetariano è carnivoro)

Si noti che questi enunciati sono diversi da quelli del tipo

Non tutti gli **S** sono **T**.

la cui rappresentazione simbolica dev'essere

$\neg \forall x(Sx \rightarrow Tx)$

'Non tutti gli Australiani sono tristi' è ben diverso da 'Nessun Australiano è triste'. Nel primo caso, infatti, è possibile che qualche Australiano sia triste se l'enunciato vale TRUE; nel secondo caso, no.

12.5 DOSSIER E MODELLI

Nell'ambito della logica enunciativa avevamo introdotto il concetto di situazione: un possibile scenario in cui alcuni enunciati valgono TRUE e altri valgono FALSE. Il mondo reale è appunto una situazione di questo tipo. Per gli scopi della logica enunciativa, è opportuno descrivere una situazione semplicemente come un elenco di enunciati atomici associati ai rispettivi valori di verità in tale situazione. Abbiamo visto come si applica CALCOLO DEL VALORE DI VERITÀ per calcolare il valore di verità di un enunciato molecolare qualsiasi a partire da queste informazioni.
Anche nella logica predicativa considereremo tali situazioni, sempre allo scopo di determinare quando un'argomentazione è valida. La definizione di validità vista in logica enunciativa si applica anche nel caso della logica predicativa: un'argomentazione è valida se e solo se la conclusione vale TRUE in tutte le situazioni in cui le premesse valgono TRUE. In logica predicativa, però, la descrizione delle situazioni deve tener conto non solo dei valori di verità degli enunciati atomici, ma anche degli individui e delle loro proprietà.
Per sviluppare un metodo sistematico di determinazione dei valori di verità di enunciati contenenti quantificatori, consideriamo dapprima il caso di una sola lettera predicativa 'F'. Diremo che un individuo *j soddisfa* il predicato 'F' in una data situazione se e solo se V(F*j*) = TRUE in tale situazione. Praticamente ciò significa che quell'individuo possiede quella proprietà in quella situazione.

INDIVIDUI RAPPRESENTATIVI

Se consideriamo un solo predicato, ogni individuo o lo soddisfa o non lo soddisfa: se il predicato 'F' significa 'è un pomodoro', allora ogni individuo o è un pomodoro o non lo è. Se consideriamo questo unico predicato, ci sono solo due tipi distinti di individui: quelli che soddisfano 'F' e quelli che non lo soddisfano. Possiamo quindi semplificare le cose, operando con due soli *individui rappresentativi*, *a* e *b*, ove *a* non soddisfa il predicato 'F', mentre *b* lo soddisfa. L'individuo *a* è rappresentativo di tutti gli individui che non soddisfano 'F', mentre l'individuo *b* è rappresentativo di tutti gli individui che soddisfano 'F'. Possiamo esprimere questo fatto mediante una tabella:

	F
a	0
b	1

Ogni riga di tale tabella sarà detta *dossier* dell'individuo indicato nella colonna di sinistra. Il dossier di un individuo indica quali predicati esso soddisfa ('1') o non soddisfa ('0'). Viceversa, ogni colonna della tabella indica quali individui soddisfano o non soddisfano il predicato riportato nell'intestazione della colonna.
Se consideriamo due predicati 'F' e 'G', vediamo che esistono quattro possibili individui rappresentativi distinti:

	F	G
a	0	0
b	0	1
c	1	0
d	1	1

È facile rendersi conto che, se i predicati distinti sono *n*, occorrono 2***n* individui rappresentativi per esaurire tutte le possibilità. Nella tabella sopra riportata non è necessario considerare un quinto individuo, poiché esso, rispetto ai predicati 'F' e 'G', si comporterà come uno dei quattro individui rappresentativi.

MODELLI

Per un dato enunciato, diremo *situazione modello*, o semplicemente *modello*, un insieme di individui e dei loro dossier. Ogni modello deve contenere almeno un individuo; inoltre, il dossier di ogni individuo deve specificare, per ogni predicato dell'enunciato, se l'individuo soddisfa quel predicato oppure no. Quando consideriamo contemporaneamente i modelli per più enunciati (ad esempio per dimostrare che un'argomentazione non è valida), i dossier degli individui devono comprendere tutti i predicati di tutti gli enunciati.
Per i due enunciati dell'argomentazione seguente:

$$\forall x(\mathrm{F}x \rightarrow \mathrm{G}x)$$
$$\therefore\ \forall y(\mathrm{G}y \rightarrow \mathrm{F}y)$$

un modello potrebbe essere:

1. Individui: *a*, *b*
2. Dossier:

	F	G
a	0	0
b	0	1

Si noti che i dossier degli individui di un modello possono essere elencati insieme, in modo da formare una tabella molto simile a una tavola di verità. Spesso descriveremo un modello fornendo semplicemente una tabella di questo tipo, in cui gli individui del modello verranno indicati nella colonna sinistra.
Il modello sopra indicato soddisfa le due condizioni precedentemente imposte: contiene almeno un individuo, e ogni dossier comprende le informazioni relative ad ogni predicato dei due enunciati originari.
Un modello può contenere un numero qualsiasi di individui. Ciò significa che per ogni enunciato esistono infiniti modelli possibili: alcuni contengono un individuo, altri ne contengono due, altri tre, eccetera. In questo paragrafo, tuttavia, ci basterà considerare solo *modelli minimali.* Un modello si dice minimale se contiene solo individui rappresentativi. Il modello appena descritto è minimale, mentre il seguente modello:

	F	G
a	0	0
b	0	1
f	0	1

non è minimale, perché gli individui *f* e *b* soddisfano esattamente gli stessi predicati: o *b* o *f* è un individuo rappresentativo, ma non tutti e due.
Vediamo un elenco completo di tutti i modelli minimali per i due enunciati sopra riportati, che contengono solo i predicati 'F' e 'G':

I.

	F	G
a	0	0

II.

	F	G
b	0	1

III.

	F	G
c	1	0

IV.

	F	G
d	1	1

V.

	F	G
a	0	0
b	0	1

VI.

	F	G
a	0	0
c	1	0

VII.

	F	G
a	0	0
d	1	1

VIII.

	F	G
b	0	1
c	1	0

IX.

	F	G
b	0	1
d	1	1

X.

	F	G
c	1	0
d	1	1

XI.

	F	G
a	0	0
b	0	1
c	1	0

XII.

	F	G
a	0	0
b	0	1
d	1	1

XIII.

	F	G
a	0	0
c	1	0
d	1	1

XIV.

	F	G
b	0	1
c	1	0
d	1	1

XV.

	F	G
a	0	0
b	0	1
c	1	0
d	1	1

I modelli possibili sono infiniti, mentre i modelli minimali sono (2***m*) – 1, ove *m* è il numero degli individui rappresentativi. In questo caso gli individui rappresentativi sono quattro; perciò i modelli minimali sono (2**4) – 1 = 15.
Abbiamo così introdotto i concetti fondamentali di modello, individuo rappresentativo e modello minimale. Dobbiamo ora definire un procedimento per determinare quando un enunciato vale TRUE in un dato modello.

DETERMINAZIONE DEL VALORE DI VERITÀ DI UN ENUNCIATO IN UN MODELLO

La determinazione, in un dato modello, del valore di verità di un enunciato contenente quantificatori non è facile come nel caso della logica enunciativa. Estenderemo e modificheremo le tecniche viste nel Capitolo 6, basate sull'ALGORITMO DI WANG. Ricordiamo che si tratta di costruire una coppia di liste: la lista sinistra contiene gli enunciati cui si vuole attribuire il valore di verità TRUE, mentre la lista destra contiene gli enunciati cui si vuole attribuire il valore di verità FALSE. Utilizziamo questo metodo per determinare il valore di verità di un singolo enunciato contenente un quantificatore.
Consideriamo, ad esempio, l'enunciato

$\forall x(\mathrm{F}x \vee \mathrm{G}x)$

nel modello:

	F	G
a	1	0
b	0	0

Come nell'ALGORITMO DI WANG, cerchiamo di attribuire il valore di verità TRUE all'enunciato dato; perciò lo mettiamo nella lista sinistra:

$\forall x(\mathrm{F}x \lor \mathrm{G}x)$ |

Quando vale TRUE, in un modello, un enunciato quantificato universalmente come questo? Solo quando tutti gli esemplari della formula non quantificata valgono TRUE. In questo modello, gli esemplari sono '$(\mathrm{F}a \lor \mathrm{G}a)$' e '$(\mathrm{F}b \lor \mathrm{G}b)$', ottenuti eliminando il quantificatore iniziale e sostituendo le occorrenze ora libere della variabile prima con '*a*' e poi con '*b*'. Mettendo entrambi questi esemplari nella lista sinistra, abbiamo:

$(\mathrm{F}a \lor \mathrm{G}a)$ |
$(\mathrm{F}b \lor \mathrm{G}b)$ |

Poiché nessuno di questi due enunciati della lista sinistra contiene un quantificatore, possiamo procedere secondo le regole già note dell'ALGORITMO DI WANG. Il primo enunciato è una disgiunzione appartenente alla lista sinistra; perciò occorre una ramificazione:

F*a*	|	G*a*	|
$(\mathrm{F}b \lor \mathrm{G}b)$	|	$(\mathrm{F}b \lor \mathrm{G}b)$	|

In entrambe le coppie di liste, il secondo enunciato di sinistra è ancora una disgiunzione; perciò occorre un'altra ramificazione per entrambe le coppie di liste:

F*a*	|	F*a*	|	G*a*	|	G*a*	|
F*b*	|	G*b*	|	F*b*	|	G*b*	|

Si osservi che le quattro coppie di liste hanno tutte la lista destra vuota. Non possiamo applicare ancora l'ALGORITMO DI WANG, perché a questo punto tutti gli enunciati delle liste sono formule predicative semplici, che non contengono né un quantificatore né un connettivo. Ricordiamo che agli enunciati delle liste di sinistra si vuole attribuire il valore di verità TRUE. Nel modello, i due enunciati della prima coppia di liste sono soddisfatti? Per rispondere a questa domanda esaminiamo i dossier. 'F' vale TRUE in questa situazione modello, ma 'F*b*' no;

perciò questa ramificazione fallisce. Possiamo indicare tale fallimento ponendo uno '0' sotto la ramificazione. Consideriamo ora la ramificazione successiva; in essa, 'F*a*' vale TRUE, ma 'G*b*' no; perciò anche questa ramificazione fallisce. Ragionando allo stesso modo per la terza e la quarta ramificazione otteniamo infine:

F*a*	F*a*	G*a*	G*a*
F*b*	G*b*	F*b*	G*b*
0	0	0	0

Tutte le ramificazioni originate dall'enunciato '∀*x*(F*x*∨G*x*)' falliscono. In virtù del fallimento di tutte le ramificazioni, possiamo concludere che l'enunciato originario non vale TRUE nel modello assegnato.
Definiamo ora un criterio per valutare le coppie di liste relative a un modello:

> L'enunciato originario vale TRUE in un modello se una delle ramificazioni provenienti da esso ha successo.

Una formulazione equivalente è:

> L'enunciato originario vale FALSE in un modello se falliscono tutte le ramificazioni provenienti da esso.

Una ramificazione ha successo quando in entrambe le liste compaiono solo formule predicative semplici e nel modello risulta soddisfatto ogni enunciato della lista sinistra, ma nessun enunciato della lista destra. Quando questa condizione è verificata, scriviamo un '1' sotto la ramificazione, per indicare che ha avuto successo. A questo punto ci possiamo fermare, perché anche una sola ramificazione che abbia avuto successo è sufficiente per dimostrare che l'enunciato originario vale TRUE nel modello.
Una ramificazione fallisce se si verifica una delle condizioni seguenti:

1. Uno stesso enunciato compare sia nella lista destra, sia nella lista sinistra.
2. In entrambe le liste compaiono solo formule predicative semplici, ma nel modello non è soddisfatta una formula della lista sinistra, oppure è soddisfatta una formula della lista destra.

Quando una ramificazione fallisce, scriviamo uno '0' sotto di essa e passiamo alla ramificazione successiva. Se non rimangono altre ramificazioni da esaminare e se tutte le ramificazioni precedenti sono fallite, concludiamo che l'enunciato originario non vale TRUE nel modello assegnato.
Ai passi dal 3(i) al 3(vii) dell'ALGORITMO DI WANG ne aggiungiamo altri quattro:

(viii) Se un enunciato è quantificato universalmente e si trova nella lista sinistra,

si cancella tale enunciato e si aggiungono alla lista sinistra tutti gli esemplari della formula, privata del suo quantificatore iniziale.

(ix) Se un enunciato è quantificato universalmente e si trova nella lista destra, si effettua una ramificazione multipla e in ogni nuova coppia di liste si sostituisce l'enunciato originario con un suo esemplare distinto.

(x) Se un enunciato è quantificato esistenzialmente e si trova nella lista sinistra, si effettua una ramificazione multipla e in ogni nuova coppia di liste si sostituisce l'enunciato originario con un suo esemplare distinto.

(xi) Se un enunciato è quantificato esistenzialmente e si trova nella lista destra, si cancella tale enunciato e si aggiungono alla lista destra tutti gli esemplari della formula, privata del suo quantificatore iniziale.

Il numero degli esemplari da aggiungere ai passi (viii) e (xi), così come il numero delle ramificazioni necessarie ai passi (ix) e (x), è pari al numero degli individui del modello. È ovvio che in ramificazioni distinte si devono usare esemplari distinti. Nell'esempio precedente c'erano solo due individui, *a* e *b*.
Usando la stessa situazione modello, consideriamo un enunciato un po' più complesso:

$$\exists x \exists y(Gx \vee \neg Fy) \mid$$

$$\exists y(Ga \vee \neg Fy) \mid \qquad\qquad \exists y(Gb \vee \neg Fy) \mid$$

$$(Ga \vee \neg Fa) \mid \quad (Ga \vee \neg Fb) \mid \qquad (Gb \vee \neg Fa) \mid \quad (Gb \vee \neg Fb) \mid$$

$$Ga \mid \quad \neg Fa \mid \quad Ga \mid \quad \neg Fb \mid \qquad Gb \mid \quad \neg Fa \mid \quad Gb \mid \quad \neg Fb \mid$$

$$0 \qquad\qquad 0 \qquad\qquad\qquad 0 \qquad\qquad 0$$

$$\mid Fa \qquad\qquad \mid Fb \qquad\qquad\qquad \mid Fa \qquad\qquad \mid Fb$$

$$0 \qquad\qquad 1 \qquad\qquad\qquad 0 \qquad\qquad 1$$

Poiché nel modello 'F*b*' non è soddisfatto, due ramificazioni hanno successo; perciò l'enunciato originario vale TRUE in questo modello. Si tenga presente che qui stiamo usando solo una parte dell'ALGORITMO DI WANG: vogliamo semplicemente determinare se un enunciato vale TRUE in un modello assegnato. È chiaro che il ricorso alle coppie di liste e alle ramificazioni è conveniente solo quando il numero di individui del modello è relativamente modesto, oppure quando l'enunciato originario contiene solo pochi quantificatori.
Occupiamoci infine della determinazione del valore di verità della premessa e della conclusione di una semplice argomentazione:

$$\forall x(Fx \rightarrow Gx)$$
$$\therefore \ \forall y(Gy \rightarrow Fy)$$

Si consideri il modello minimale 1:

	F	G
a	0	0

Applicando alla premessa il metodo delle coppie di liste, otteniamo:

$\forall x(Fx \rightarrow Gx) |$

$(Fa \rightarrow Ga) |$

$\neg Fa |$ $\qquad\qquad$ $Ga |$

$\qquad\qquad\qquad\qquad$ 0

$| Fa$

1

Poiché una delle ramificazioni si conclude con un '1', in questo modello la premessa vale TRUE. Applicando lo stesso metodo alla conclusione, scopriamo che anch'essa vale TRUE in questo modello. Tuttavia, non possiamo ancora concludere che l'argomentazione sia valida, perché dobbiamo considerare *tutti* i modelli minimali: non deve esistere nessun modello in cui le premesse valgano TRUE e la conclusione valga FALSE, altrimenti l'argomentazione non è valida. Consideriamo dunque questa argomentazione relativamente al modello minimale 2:

	F	G
b	0	1

Applicando il metodo delle coppie di liste, si scopre che in questo modello la premessa vale TRUE, ma la conclusione vale FALSE; perciò, questo modello dimostra che l'argomentazione non è valida.

Vediamo un abbozzo di procedimento per determinare se un'argomentazione contenente quantificatori è valida:

1. INPUT gli enunciati dell'argomentazione.
2. Determinare gli individui rappresentativi in base al numero di predicati distinti contenuti nelle premesse e nella conclusione.
3. Costruire tutti i possibili modelli minimali.
4. FOR ogni modello minimale
   a. Applicare il metodo delle coppie di liste per determinare il valore di verità delle premesse e della conclusione nel modello.
   b. IF nel modello tutte le premesse valgono TRUE e la conclusione vale FALSE
      THEN OUTPUT "Argomentazione non valida" e STOP.
5. OUTPUT "Argomentazione valida" e STOP.

Questo procedimento non è algoritmico perché, in molti casi, non è possibile portare a termine i passi 2 e 3.
Restano da considerare altre due circostanze speciali. In un enunciato contenente un quantificatore può comparire anche una lettera proposizionale singola; è il caso, ad esempio, di '(A∧∃*x*F*x*)'. Quando un enunciato contiene lettere proposizionali, il modello deve anche specificare i valori di verità di questi enunciati atomici. In secondo luogo, abbiamo rappresentato gli individui dei modelli con lettere minuscole corsive; in tal modo essi potrebbero essere confusi con le occorrenze di costanti individuali negli enunciati. Converremo quindi di identificare gli individui del modello con lettere minuscole corsive diverse da qualsiasi costante individuale che compaia nell'enunciato in esame. Inoltre, ogni volta che, in una coppia di liste, compare un enunciato che contiene una costante individuale che non rientra nel campo di nessun quantificatore, tratteremo tale enunciato come se fosse quantificato esistenzialmente. Perciò, se tale enunciato compare nella lista sinistra, si effettua una ramificazione e si sostituiscono tutte le occorrenze della costante individuale con i nomi degli individui del modello; se invece tale enunciato compare nella lista destra, si cancella l'enunciato e si aggiungono tutti gli esemplari ottenuti sostituendo la costante individuale con i nomi degli individui del modello. Ad esempio,

(G*a*→∀*x*F*x*) |

(G*b*→∀*x*F*x*) | (G*c*→∀*x*F*x*) |

dove *b* e *c* sono i due individui del modello.

LIMITI DEI MODELLI

La determinazione della validità di un'argomentazione può richiedere molto tempo, se viene effettuata mediante l'esame dei modelli. In alcuni casi può capitare di trovare abbastanza rapidamente un modello in cui le premesse valgono TRUE ma la conclusione vale FALSE; in altri casi, invece, può rendersi necessario l'esame di molti modelli. In linea di principio, bisogna esaminare tutti i modelli, anche se, in realtà, abbiamo semplificato questo compito, limitandoci ai modelli minimali; nonostante questa restrizione, tuttavia, la determinazione della validità può risultare un compito gravoso. Ad esempio, nel caso di un'argomentazione contenente solo tre predicati, gli individui rappresentativi sono otto, e quindi i modelli minimali sono 255. La determinazione dei valori di verità delle premesse e della conclusione richiederebbe perciò moltissimo tempo, a meno che il compito non sia demandato a un calcolatore appositamente programmato.
Inoltre, come vedremo tra breve, anche una piccola estensione della logica predicativa, che includa le relazioni fra individui oppure la particolare relazione logica di identità (rappresentata con '=') rende pressoché inutile il concetto di modello minimale. In generale, in questi casi saremo costretti a considerare un numero in-

finito di modelli minimali.
L'esame di tutti i modelli minimali non è il modo più efficiente per determinare i valori di verità. Esistono numerose scorciatoie, troppo sofisticate per i propositi di questo libro, che permettono di convertire gli enunciati in una forma normale e di determinare poi rapidamente se la forma normale vale TRUE in un modello minimale. Riprenderemo brevemente questo argomento nel Capitolo 14.

12.6 RELAZIONI

Prima di affrontare il difficile problema di rappresentare in forma simbolica enunciati italiani più complessi usando i quantificatori, conviene notare che non tutte le argomentazioni di interesse coinvolgono individui e loro proprietà: alcune argomentazioni dipendono da relazioni fra individui; si consideri, ad esempio, l'argomentazione seguente:

Davide è più vecchio di Giovanni.
Giovanni è più vecchio di Sandro.
∴ Davide è più vecchio di Sandro.

In questo caso, la validità dell'argomentazione dipende dalle relazioni fra gli individui Davide, Giovanni e Sandro, non dalle loro proprietà. Questa argomentazione ci appare sensata e manifestamente valida, perché conosciamo il significato della relazione "essere più vecchio di" che lega due individui.
Per trattare relazioni fra due o più individui estenderemo semplicemente l'uso delle lettere predicative. Indichiamo con 'E' la relazione "essere più vecchio di" e scriviamo i nomi degli individui a destra della lettera predicativa, nell'ordine opportuno. Perciò, se

a = Davide
b = Giovanni
c = Sandro

scriveremo

E*ab*	per rappresentare	Davide è più vecchio di Giovanni.
E*bc*	per rappresentare	Giovanni è più vecchio di Sandro.
E*ac*	per rappresentare	Davide è più vecchio di Sandro.

L'ordine delle costanti individuali è importante: 'E*ba*' rappresenta 'Giovanni è più vecchio di Davide', cioè un enunciato ben diverso da 'Davide è più vecchio di Giovanni'.
Le formule ben formate e le quantificazioni che coinvolgono relazioni possono essere trattate nel modo già visto. Le lettere relazionali sono semplicemente predi-

cati a più argomenti, che devono essere seguiti da più di una costante, o da più di una variabile, per dar luogo a formule ben formate. È semplice rivedere la definizione di fbf per trattare predicati a *n* argomenti; dovremo però fare molta attenzione alle posizioni delle variabili. La fbf 'E*xb*' (*x* è più vecchio di Giovanni) può essere quantificata esistenzialmente così:

∃*x*E*xb*	Qualcuno è più vecchio di Giovanni.

Si osservi che

∃*x*E*bx*	Giovanni è più vecchio di qualcuno.

è un enunciato ben diverso.
Per tenere ben distinti gli argomenti di una relazione in un enunciato completamente quantificato, useremo variabili diverse per rappresentare individui non specificati che possano essere diversi fra loro:

∃*x*∃*y*E*xy*	Qualcuno è più vecchio di qualcuno.

Se usassimo una sola variabile, avremmo

∃*x*E*xx*	Qualcuno è più vecchio di se stesso.

enunciato il cui valore di verità è ovviamente FALSE.
Il quantificatore universale si usa in modo simile:

∀*x*E*xb*	Tutti sono più vecchi di Giovanni.

In logica, 'tutti' è inteso in modo strettamente letterale: non significa "tutti gli altri". Questo enunciato vale perciò FALSE, perché Giovanni non è più vecchio di se stesso. Allo stesso modo,

∀*x*E*bx*	Giovanni è più vecchio di tutti.

vale FALSE, perché Giovanni non è più vecchio di se stesso.
Combinando infine i quantificatori, otteniamo:

∃*x*∀*y*E*xy*	Qualcuno è più vecchio di tutti.
∀*x*∃*z*E*xz*	Tutti sono più vecchi di qualcuno.

Entrambi questi enunciati valgono FALSE nel mondo reale, perché nessuno è più vecchio di se stesso ed esiste un numero finito di individui.
Per costruire modelli per enunciati contenenti predicati relazionali, dobbiamo estendere il concetto di dossier di un individuo. Consideriamo dapprima solo un predicato a due argomenti 'R' e, per semplicità, tre individui *a*, *b* e *c*. Esistono

nove modi di correlare *a*, *b* e *c* mediante la relazione R, cioè vi sono nove enunciati di cui dobbiamo specificare il valore di verità; tali enunciati possono essere ordinati in una matrice:

R*aa*	R*ba*	R*ca*
R*ab*	R*bb*	R*cb*
R*ac*	R*bc*	R*cc*

Questa rappresentazione non è immediatamente utilizzabile per rappresentare dei dossier; con qualche ridondanza, però, possiamo costruire un dossier per ogni individuo. Il dossier di '*a*' deve illustrare quali predicati a due argomenti sono soddisfatti quando '*a*' è il primo termine e quali sono soddisfatti quando '*a*' è il secondo termine. Riguardo all'individuo '*a*' vogliamo insomma sapere quali esemplari di 'R*ax*' e quali esemplari di 'R*xa*' valgono TRUE. Per questo semplice esempio useremo quindi sei colonne, attribuendo nell'ordine i valori '*a*', '*b*' e '*c*' prima al primo individuo e poi al secondo:

	R*a*_	R*b*_	R*c*_	R_*a*	R_*b*	R_*c*
a						
b						
c						

La prima riga contiene i valori di verità degli enunciati ottenuti mettendo '*a*' al posto dello spazio lasciato vuoto nell'intestazione della corrispondente colonna; per la seconda riga si mette '*b*' negli spazi vuoti e per la terza si mette '*c*'. Se 'R*aa*' vale TRUE, allora la prima e la quarta casella della prima riga contengono un '1'; se invece 'R*aa*' non vale TRUE, queste due caselle contengono uno '0'. Per V(R*aa*) = TRUE si ha:

	R*a*_	R*b*_	R*c*_	R_*a*	R_*b*	R_*c*
a	1			1		
b						
c						

In parole povere, per costruire il dossier di *a* ci si sposta lungo la prima riga della tabella, inserendo '*a*' in ogni spazio vuoto e controllando che, quando nelle intestazioni di due colonne si ottiene la stessa espressione, figurino gli stessi valori di verità nelle caselle corrispondenti della riga appropriata.
Occorre fare attenzione a quest'ultima condizione. Si osservi che, quando si costruisce il dossier di *a*, l'intestazione della seconda colonna diventa 'R*ba*'; ma

'R*ba*' compare anche quando si costruisce il dossier di *b*, in corrispondenza della quarta colonna. Ebbene, il valore di verità di 'R*ba*' deve essere lo stesso nel dossier di *a* e nel dossier di *b*.

	R*a*_	R*b*_	R*c*_	R_*a*	R_*b*	R_*c*
a		0				
b				0		
c						

Una volta definiti i dossier, i modelli si possono costruire come avevamo fatto prima, e si può determinare il valore di verità di vari enunciati quantificati in diversi modelli.
Di solito, per costruire opportuni modelli minimali per relazioni a due argomenti occorre un gran numero di individui. Se l'universo del discorso contiene un numero di individui appena più che irrisorio, diventa molto difficile rappresentare e usare i dossier e le tabelle. Il concetto di dossier aiuta tuttavia a capire come si deve valutare un enunciato quantificato, anche se il dossier non è facile da rappresentare.

12.7 RAPPRESENTAZIONE SIMBOLICA DI ENUNCIATI

Per rappresentare simbolicamente un enunciato espresso in italiano occorre capire bene le condizioni di verità dell'enunciato originario. Ad esempio, se vogliamo affermare che

Solo agli studenti piace divertirsi.

dobbiamo prima capire bene in quali condizioni questo enunciato vale TRUE e in quali vale FALSE. Se si scoprisse che a qualche persona che non è uno studente (ad esempio, a un insegnante) piace divertirsi, l'enunciato avrebbe valore di verità FALSE. In altre parole, l'enunciato afferma che tutte le persone a cui piace divertirsi sono studenti. La rappresentazione simbolica è allora immediata:

$\forall x(\mathrm{D}x \rightarrow \mathrm{E}x)$

Un enunciato come

In marzo piove o nevica ogni giorno.

si può rappresentare simbolicamente come:

$\forall x(Lx \rightarrow (Ax \text{ XOR } Nx))$

cioè per ogni individuo x, se x è un giorno di marzo, allora x è piovoso o è nevoso.
Occorre più attenzione nel caso di enunciati come

I pipistrelli e i gatti sono mammiferi.

La rappresentazione simbolica corretta è

$(\forall x(Ix \rightarrow Mx) \wedge \forall x(Gx \rightarrow Mx))$

oppure

$\forall x((Ix \vee Gx) \rightarrow Mx)$

Se si trascurassero le condizioni di verità dell'enunciato originario, si potrebbe rappresentare scorrettamente l'enunciato mediante una formula che inizia con

Per ogni x, se x è un pipistrello *e* x è un gatto...

Ma non esiste niente che sia allo stesso tempo un pipistrello e un gatto, ed è evidente che l'enunciato originario non parla di ipotetici pipistrelli-gatti.
Se si introducono i quantificatori e i predicati, si possono esprimere in forma simbolica enunciati di complessità sempre maggiore. Ci si può però trovare di fronte ad enunciati ambigui. Se diciamo

Dio aiuta chi si aiuta.

potremmo intendere

1. Dio aiuta *solo* quelli che si aiutano.

oppure

2. Dio aiuta *tutti* quelli che si aiutano.

oppure l'enunciato potrebbe avere entrambi i significati.
Rappresentando 'x aiuta x' (cioè x aiuta se stesso) con 'Axx' e 'Dio aiuta x con 'Adx', otterremo due rappresentazioni simboliche diverse, a seconda dell'interpretazione:

1'. $\forall x(Adx \rightarrow Axx)$

e

2′. $\forall x(Axx \rightarrow Adx)$

Spesso risulterà utile costruire la rappresentazione simbolica di un enunciato complesso in più passaggi, a partire dall'inizio dell'enunciato. Si consideri questo esempio:

3. Tutti gli studenti ammirano qualche professore.

Poiché questa è un'affermazione universale riguardante tutti gli studenti, la prima fase dalla rappresentazione simbolica consiste nell'evidenziare la quantificazione universale:

$\forall x$(se x è uno studente, allora x ammira qualche professore)

Rappresentiamo ora 'x è uno studente' con 'Ex', ottenendo:

$\forall x(Ex \rightarrow x$ ammira qualche professore)

Per rappresentare 'x ammira qualche professore', indichiamo 'x ammira y' con 'Axy' e 'y è un professore' con 'Fy':

$\exists y(Fy \wedge Axy)$

Usiamo ora questo enunciato come conseguente del condizionale, al posto di 'x ammira qualche professore':

3. $\forall x(Ex \rightarrow \exists y(Fy \wedge Axy))$

In questo caso, abbiamo usato 'Fy' per rappresentare 'y è un professore'. Avremmo però potuto benissimo usare una variabile diversa da y (ad esempio, z) in entrambe le posizioni, ottenendo una rappresentazione simbolica ugualmente corretta:

3a. $\forall x(Ex \rightarrow \exists z(Fz \wedge Axy))$

Consideriamo ora un esempio analogo:

4. I professori invidiano qualunque professore che sia ammirato dagli studenti.

In questo caso abbiamo un'affermazione universale a proposito dei professori. Il primo passaggio ci condurrà a:

$\forall x(Fx \rightarrow x$ invidia qualunque professore che sia ammirato dagli studenti)

Dobbiamo ora caratterizzare un generico professore che sia ammirato dagli studenti e indicare che x invidia un tale professore:

Per ogni y, se y è un professore ammirato dagli studenti, allora x invidia y.

'y è un professore ammirato da (alcuni) studenti' si può rappresentare così:

$(\mathrm{F}y \land \exists z(\mathrm{E}z \land \mathrm{A}zy))$

Mettendo insieme questi frammenti, otteniamo:

4. $\forall x(\mathrm{F}x \rightarrow \forall y((\mathrm{F}y \land \exists z(\mathrm{E}z \land \mathrm{A}zy)) \rightarrow \mathrm{I}xy))$

12.8 CONCLUSIONI

La logica predicativa ci porta al di là del semplice assegnamento di valori di verità ad enunciati, richiedendoci di prendere in considerazione gli individui e le loro proprietà, la quantificazione e le relazioni fra individui. Gli individui sono rappresentati da lettere minuscole corsive dell'inizio dell'alfabeto, mentre le variabili individuali sono lettere minuscole corsive della fine dell'alfabeto. Le lettere predicative sono lettere maiuscole dell'inizio dell'alfabeto. Un predicato a n argomenti ($n \geqslant 1$) seguito da n costanti individuali è un enunciato.
Fra le possibili stringhe (sequenze di simboli) abbiamo distinto le formule ben formate (fbf). Una fbf preceduta da un quantificatore è ancora una fbf. I quantificatori sono o universali o esistenziali; hanno un campo di applicazione, e ogni occorrenza della variabile del quantificatore entro il campo del quantificatore stesso si dice legata. Una fbf priva di occorrenze libere di variabili è un enunciato.
Le condizioni di verità di un enunciato quantificato sono:

$\vee(\forall x\mathrm{S}x)$ = TRUE se e solo se ogni esemplare di '$\mathrm{S}x$' ha valore di verità TRUE.
$\vee(\exists x\mathrm{S}x)$ = TRUE se e solo se almeno un esemplare di '$\mathrm{S}x$' ha valore di verità TRUE.

Abbiamo poi introdotto i concetti di modello, di modello minimale e di dossier di un individuo. Infine, abbiamo formulato alcuni consigli per la rappresentazione simbolica di enunciati espressi in italiano.

12.9 ESERCIZI

A. Usando le costanti individuali e le lettere predicative sotto indicate, rappresentare simbolicamente gli enunciati seguenti.

a = Socrate	F = è (o era) un filosofo
b = Platone	G = è (o era) uno studioso di logica

1. Socrate è uno studioso di logica.
2. Platone era uno studioso di logica.
3. Socrate era un filosofo e Platone era uno studioso di logica.
4. Platone non era uno studioso di logica, ma Socrate era un filosofo.
5. Se Platone era un filosofo, allora lo era anche Socrate.

B. Usando le corrispondenze dell'Esercizio A, esprimere in italiano le formule seguenti:

1. $\mathrm{F}a$
2. $\mathrm{F}b$
3. $(\mathrm{G}a \vee \mathrm{F}b)$
4. $(\mathrm{G}b \wedge \neg \mathrm{F}a)$
5. $(\mathrm{G}a \leftrightarrow \mathrm{G}b)$

C. Posto C = è un programmatore di calcolatori
D = è un filosofo
E = è un matematico
rappresentare in forma simbolica gli enunciati seguenti:

1. Ognuno è un filosofo.
2. Qualcuno è un matematico.
3. Nessuno è un programmatore di calcolatori.
4. Non tutti sono matematici.
5. Tutti i matematici sono filosofi.
6. Alcuni filosofi sono programmatori di calcolatori.
7. Ogni programmatore di calcolatori non è un filosofo.
8. Alcuni filosofi non sono programmatori di calcolatori.
9. Non tutti i matematici sono filosofi.
10. Ognuno è o un matematico o un filosofo.
11. Ognuno è un matematico e un filosofo.
12. Nessun matematico programma calcolatori.

D. Usando le corrispondenze dell'Esercizio C, esprimere in italiano le formule seguenti:

1. $\forall x Cx$
2. $\forall y Cy$
3. $\exists u Du$
4. $\neg \exists x Dx$
5. $\exists x \neg Dx$
6. $\forall x \neg Ex$
7. $\forall x(Cx \rightarrow Dx)$
8. $\exists x(Cx \rightarrow Dx)$
9. $\exists x(Cx \wedge Ex)$
10. $\forall x(Cx \wedge Ex)$

E. Si considerino i quindici modelli minimali elencati in questo capitolo. In quali di essi valgono TRUE i seguenti enunciati? In quali valgono FALSE?

1. $\forall x Fx$
2. $\forall x(Fx \rightarrow Gx)$
3. $\forall x(Fx \vee Gx)$
4. $\forall x(Fx \wedge Gx)$
5. $\forall x \neg Gx$
6. $\exists x Fx$
7. $\neg \exists x \neg Gx$
8. $\exists x \neg Gx$
9. $\neg \forall x Gx$
10. $\exists x(Fx \vee Gx)$
11. $\exists x(Fx \wedge Gx)$
12. $\exists x(Fx \rightarrow Gx)$
13. $\forall x(Fx \vee Fx)$
14. $\forall x(Fx \rightarrow Fx)$
15. $\exists x(Fx \wedge Fx)$

F. Posto L = è più alto di
a = Andrea
b = Beatrice

rappresentare in forma simbolica gli enunciati seguenti:

1. Andrea è più alto di Beatrice.
2. Beatrice è più alta di Andrea.
3. Andrea non è più alto di se stesso.
4. Qualcuno è più alto di Beatrice.
5. Andrea è più alto di tutti.
6. Ognuno è più alto di qualcuno. (Attenzione!)
7. Qualcuno è più alto di tutti.

G. Usando le stesse corrispondenze dell'Esercizio F, esprimere in italiano gli enunciati seguenti:

1. Lba
2. $\neg Lab$
3. $\neg Lbb$
4. $\forall x Lax$
5. $\forall x Lxa$
6. $\exists x Lbx$
7. $\exists x Lxb$
8. $\exists x \forall y Lxy$
9. $\exists x \forall y Lyx$
10. $\forall x \exists y Lxy$
11. $\forall x \exists y Lyx$
12. $\forall x Lxx$
13. $\exists y Lyy$

H. Rappresentare in forma simbolica gli enunciati seguenti:
1. Se Sandro ama davvero Maria, allora nessuno ama Sandro.
2. Tutti i possessori di fucili violano qualche legge. (Si ponga Pxy = x possiede y, Fx = x è un fucile, Vxy = x viola y, Lx = x è una legge.)
3. Tutti gli amanti degli amanti amano se stessi. (Si usi solo il predicato Axy = x ama y.)
4. Ogni dottore che cura se stesso ha per paziente uno squilibrato. (Si ponga Cxy = x cura y, Sx = x è uno squilibrato, Px = x è un paziente.)
5. Alcuni uomini d'affari hanno le mani in pasta in ogni affare. (Si ponga Ax = x è un uomo d'affari, $Hxyz$ = x ha y in z, Mxy = x è una mano di y, Fx = x è un affare.)
6. Tutti quelli che ammirano se stessi e che non amano nessun altro al di fuori della propria madre verranno eletti in ogni paese. (Si ponga Axy = x ammira y, $A1xy$ = x ama y, Mxy = x è la madre di y, Exy = x verrà eletto in y, Px = x è un paese).
7. A nessuna persona che disprezzi Napoleone piacciono le barzellette sporche.
8. Nessuno ammira qualcuno che cerca di fare tutto.
9. Tutti hanno qualche problema, ma nessuno ha tutti i problemi.
10. Ciascuno o ammira o disprezza Garibaldi.
11. Se qualcuno mi farà un regalo stupido per il mio compleanno, non gli manderò un bigliettino di ringraziamento. (Si ponga $Dxyzw$ = x dà y a z in occasione di w, Rx = x è un regalo stupido, Cx = x è il mio compleanno, m = io/me, $Mxyz$ = x manda a y una z, Bx = x è un bigliettino di ringraziamento.)

12.10 SUGGERIMENTI PER UNA REALIZZAZIONE SU CALCOLATORE

Non è molto difficile trasformare in un programma per calcolatore l'algoritmo per decidere quando è un enunciato ben formato una stringa che può contenere quantificatori, variabili o costanti: dobbiamo solo modificare l'algoritmo definito nel Capitolo 5. Siccome pochi calcolatori sono dotati dei simboli '∀' e '∃', si possono usare le lettere 'A' ed 'E' per rappresentarli, evitando di usarle per indicare enunciati o predicati; in alternativa si potrebbero usare le parole 'TUTTI' e 'ALCUNI'.

Se ci limitiamo ad enunciati contenenti predicati ad un argomento, possiamo anche scrivere un programma che determini se un'argomentazione è valida o no. In questo capitolo abbiamo presentato un abbozzo del relativo algoritmo; per scrivere il programma corrispondente, occorrerà innanzitutto definire una struttura di dati in cui memorizzare uno o più modelli. Come abbiamo suggerito, un modello può essere memorizzato sotto forma di una matrice, le cui righe (dossier) indicano i predicati soddisfatti da diversi individui. Si dovranno poi contare i predicati distinti contenuti negli enunciati dell'argomentazione, per poter definire tutti gli individui rappresentativi. Per quest'ultima operazione, ci si può ispirare al modo in cui, nel Capitolo 6, sono state generate tutte le possibili combinazioni di valori di verità; inoltre, se gli individui rappresentativi sono identificati mediante numeri, possono essere associati direttamente ai propri dossier. Occorre poi produrre tutti i modelli minimali; ciò equivale a produrre tutte le varie combinazioni di individui rappresentativi.

Dopo aver prodotto i modelli minimali, dobbiamo definire un procedimento per determinare i valori di verità delle premesse e della conclusione in un modello assegnato. Come abbiamo visto, a questo scopo si può utilizzare il metodo su cui si fonda l'ALGORITMO DI WANG (vedi passo 3 al Capitolo 6), aggiungendo le condizioni relative al caso in cui il "connettivo principale" sia un quantificatore. Per la realizzazione di questi passi mediante un programma, si vedano i suggerimenti per la realizzazione su calcolatore forniti nel Capitolo 6. Si noti che occorre anche un sottoprogramma VERIFICARE leggermente diverso da quello definito nel Capitolo 6 per l'ALGORITMO DI WANG.

12.11 ESERCIZI DI PROGRAMMAZIONE

1. Scrivere un algoritmo (o un programma) che riceva in ingresso gli enunciati di un'argomentazione e determini il numero totale di predicati distinti in essi contenuti.
2. Scrivere un algoritmo (o un programma) che riceva in ingresso una lista di predicati distinti, eventualmente prodotta da un apposito sottoprogramma, e generi tutti gli individui rappresentativi.
3. Modificare l'algoritmo CONNETTIVO PRINCIPALE del Capitolo 6 in mo-

do che, se una formula è quantificata universalmente o esistenzialmente, venga considerato "connettivo principale" il quantificatore il cui campo copre il resto della formula. Il quantificatore iniziale non è propriamente un connettivo, perché non connette fra loro due enunciati; è un "connettivo" un po' nel modo in cui lo è la negazione.

4. Scrivere un algoritmo (o un programma) che, ricevendo in ingresso una formula quantificata universalmente o esistenzialmente e una lista di individui, effettui le operazioni seguenti; (a) cancelli il quantificatore iniziale; (b) fornisca in uscita tutti gli esemplari della formula così ottenuta.
5. Scrivere un algoritmo (o un programma) per determinare se una formula predicativa è semplice, cioè contiene solo un predicato e delle costanti individuali.

13

Logica predicativa: regole di inferenza per i quantificatori

Ricordiamo che un'argomentazione si dice valida se è impossibile che la conclusione valga FALSE quando tutte le premesse valgono TRUE. In logica enunciativa esistono diversi modi per determinare se un'argomentazione è valida o no; ad esempio, si può costruire una tavola di verità ed esaminare tutte le situazioni, per vedere se è possibile che le premesse valgano TRUE e la conclusione valga FALSE. In logica predicativa, invece, in generale non è possibile costruire o esaminare tutti i modelli, compresi quelli in cui le premesse sono vere o la conclusione è falsa. Di conseguenza, occorre trovare altri modi per dimostrare la validità di un'argomentazione. Uno dei modi più semplici consiste nel derivare la conclusione dalle premesse in un sistema formale di deduzione, per mezzo di regole di inferenza che conservino la verità. Il sistema deve essere completo, nel senso che nel sistema deve essere possibile derivare ogni conclusione di un'argomentazione valida.
Ci poniamo ora il problema di derivare conclusioni nel caso di enunciati contenenti quantificatori e variabili. L'uso di quantificatori e di variabili per rappresentare la struttura degli enunciati ha lo scopo di permetterci di derivare conclusioni che non potremmo dimostrare con i metodi della sola logica enunciativa. Ci sentiamo sicuri del fatto che l'argomentazione

1. Tutti i Greci sono mortali.
2. Socrate è Greco.

∴ 3. Socrate è mortale.

sia valida; dobbiamo però dimostrarlo, e dobbiamo anche definire delle regole che ci permettano di derivare la conclusione dalle premesse.
Questa argomentazione può essere rappresentata simbolicamente nel modo seguente:

1. $\forall x(\mathrm{G}x \rightarrow \mathrm{M}x)$
2. $\mathrm{G}c$

∴ 3. $\mathrm{M}c$

L'enunciato (1) significa che, per ogni individuo x, '(Gx→Mx)' è soddisfatto da quell'individuo. Perciò un esemplare che usi 'c' (cioè la costante che rappresenta Socrate) al posto di 'x' vale TRUE. In altre parole, se (1) vale TRUE, allora vale TRUE anche

1a. (Gc→Mc)

A questo punto possiamo applicare →ELIM a (1a) e (2) per ottenere (3).
Le regole di introduzione e di eliminazione dei connettivi viste in logica enunciativa rimangono invariate; dobbiamo solo aggiungere alcune regole per introdurre ed eliminare i quantificatori. La strategia generale consisterà nell'eliminare i quantificatori in qualche modo, per poi manipolare e trasformare i risultati secondo le regole già note dalla logica enunciativa, ed introdurre infine, se necessario, i quantificatori appropriati per ottenere la conclusione desiderata. Queste nuove regole per l'introduzione e l'eliminazione dei quantificatori sono espresse in modo assai preciso, ed occorre fare molta attenzione non solo all'enunciato della riga cui si applica la regola, ma anche ad altri enunciati della prova o della sottoprova.

13.1 REGOLE PER LA QUANTIFICAZIONE UNIVERSALE

Le regole di inferenza dei Capitoli 8 e 9 si applicano ad enunciati quantificati considerati come atomici. Ad esempio, ∧ELIM si può applicare ad una riga contenente l'enunciato '(∀xFx∧∀yGy)': consideriamo '∀xFx' e '∀yGy' come due enunciati singoli **P** e **Q**, cioè consideriamo '(∀xFx∧∀yGy)' come un enunciato avente la forma (**P**∧**Q**). Ora però estenderemo il nostro sistema formale di deduzione in modo da permettere anche derivazioni che prendano direttamente in considerazione i quantificatori.

ELIMINAZIONE DEL QUANTIFICATORE UNIVERSALE

Se riflettiamo sulle condizioni di verità di un enunciato quantificato universalmente, come '∀xFx', notiamo che esso vale TRUE solo se tutti gli esemplari di 'Fx' valgono TRUE. Perciò, inferendo da '∀xFx' un esemplare qualsiasi, non passeremo mai da un enunciato vero a uno falso. Ciò giustifica la regola

∀ELIM: Da un enunciato del tipo
∀vP
si può derivare
P[c/v]

Nella formulazione della regola, '**v**' rappresenta una variabile generica (w, x, y, z, ...), mentre '**c**' rappresenta una costante generica (a, b, c, d, ...). Con **P[c/v]**

indichiamo il risultato della sostituzione di tutte le occorrenze libere della variabile **v** nella formula **P** con la costante **c**.
Ad esempio, se **P** è $\exists x(Fx \lor Gy)$', allora **P**[a/y] è '$\exists x(Fx \lor Ga)$', mentre **P**[a/x] è ancora '$\exists x(Fx \lor Gy)$', perché 'x' non è libera in **P**.
In una derivazione, l'uso di $\forall$ELIM è del tipo seguente:

10. $\forall x(Fx \rightarrow Hx)$: <PREMESSA o regola>

.
.
.

15. $(Fd \rightarrow Hd)$: $\forall$ELIM,10

L'enunciato della riga 10 è quantificato universalmente; l'enunciato della riga 15 si ottiene da quello della riga 10 eliminando il quantificatore iniziale e sostituendo tutte le occorrenze ora libere della variabile 'x' del quantificatore con la costante individuale 'd'.
È essenziale notare che questa regola e tutte le altre regole per i quantificatori richiedono che il campo del quantificatore iniziale si estenda sino alla fine dell'enunciato contenuto nella riga citata. Vediamo un esempio di enunciato cui non si può applicare la regola $\forall$ELIM, perché il campo dell'espressione quantificatrice universale '$\forall x$' non è l'intero enunciato:

$(\forall xFx \lor \forall yGy)$

Da questo enunciato non si può ricavare '$(Fa \lor \forall yGy)$' mediante $\forall$ELIM.
$\forall$ELIM si può usare più volte facendo riferimento sempre alla stessa riga:

10. $\forall x(Fx \rightarrow Hx)$

.
.
.

15. $(Fd \rightarrow Hd)$: $\forall$ELIM,10
16. $(Fe \rightarrow He)$: $\forall$ELIM,10

INTRODUZIONE DEL QUANTIFICATORE UNIVERSALE

È anche utile essere in grado di generalizzare, cioè di introdurre, un quantificatore universale. Per giustificare una regola di questo tipo ci si può rifare alle entità geometriche elementari: quando un insegnante disegna alla lavagna un triangolo e poi usa quello specifico triangolo per dimostrare dei teoremi riguardanti tutti i triangoli,ciò ha un senso purché egli non faccia riferimento a nessuna proprietà specifica del triangolo disegnato come esempio. In altre parole, se siamo in grado di dimostrare qualcosa per un individuo scelto arbitrariamente, la stessa cosa si può considerare dimostrata per qualsiasi individuo; dobbiamo solo accertarci che le particolari proprietà dell'individuo scelto non svolgano alcun ruolo nella prova.

La regola seguente è appunto definita in modo da assicurare che questa condizione sia verificata.

∀INTR: Da un enunciato
P
si può derivare
∀vP[v/c]
purché:
1. **c** non compaia in nessuna premessa.
2. Se **P** compare in una sottoprova, nessuna costante di **P** compaia in un'ASSUNZIONE tuttora in vigore.
3. Tutte le nuove occorrenze della variabile **v** in **P** siano libere dopo la sostituzione **P[v/c]**.

A proposito della condizione (2), un'ASSUNZIONE è considerata "in vigore" nella sottoprova che la segue e in ogni sotto-sottoprova in essa contenuta. Infine, la notazione **P[v/c]** significa che tutte le occorrenze della costante **c** vengono sostituite dalla variabile **v**; la condizione (3) significa perciò che, quando **v** sostituisce **c**, non deve essere interna al campo di applicazione di un quantificatore già presente che contenga **v**: la nuova occorrenza della variabile non deve, per così dire, essere "catturata" da un quantificatore già presente nella fbf **P**.
Di seguito sono riportati alcuni esempi di uso corretto di ∀INTR; in ognuno di essi si supponga che per la costante della riga 5 siano verificate le restrizioni ora specificate.

Esempio 1.
5. $(Fa \rightarrow Ga)$: <regola>
.
.
.
9. $\forall x(Fx \rightarrow Gx)$: ∀INTR,5

Esempio 2.
5. $(Fb \vee \exists y Gy)$: <regola>
.
.
.
12. $\forall x(Fx \vee \exists y Gy)$: ∀INTR,5

Vediamo una derivazione completa per l'argomentazione seguente:

$\forall x(Fx \rightarrow Gx)$
∴ $(\forall y Fy \rightarrow \forall z Gz)$

1. $\forall x(Fx \rightarrow Gx)$: PREMESSA
/INIZIO: $\rightarrow$INTR per derivare $(\forall yFy \rightarrow \forall zGz)$/
*2. $\forall yFy$: ASSUNZIONE
*3. Fa : $\forall$ELIM,2
*4. $\forall x(Fx \rightarrow Gx)$: INVIATO,1
*5. $(Fa \rightarrow Ga)$: $\forall$ELIM,4
*6. Ga : $\rightarrow$ELIM,5,3
*7. $\forall zGz$: $\forall$INTR,6
*8. $(\forall yFy \rightarrow \forall zGz)$: $\rightarrow$INTR,2,7
/FINE: $\rightarrow$INTR per derivare $(\forall yFy \rightarrow \forall zGz)$/
9. $(\forall yFy \rightarrow \forall zGz)$: RESTITUITO,8

Vediamo un esempio di uso scorretto di $\forall$INTR:

Esempio 3.
5. $(Fa \rightarrow Ga)$: <regola>
.
.
.
9. $\forall x(Fx \rightarrow Ga)$: $\forall$INTR,5 [scorretto: non sono state sostituite tutte le occorrenze di '*a*']

Un altro esempio di uso scorretto di $\forall$INTR è:

Esempio 4.
5. $(Fa \rightarrow \exists x(Ga \wedge Hx))$: <regola>
.
.
.
9. $\forall x(Fx \rightarrow \exists x(Gx \wedge Hx))$: $\forall$INTR,5 [scorretto: la '*x*' di 'G*x*' è stata "catturata"]

Si osservi che '*x*', avendo sostituito la '*a*' di 'G*a*' della riga 5, è stata catturata dal quantificatore esistenziale già presente. Invece di '*x*' potremmo usare un'altra variabile, ad esempio '*y*', per inferire correttamente:

9. $\forall y(Fy \rightarrow \exists x(Gy \wedge Hx))$: $\forall$INTR,5

Con queste due regole possiamo derivare la conclusione di alcune argomentazioni tradizionalmente studiate fin dai tempi di Aristotele. Una, ad esempio, è l'antica argomentazione sillogistica:

Tutti gli uomini sono mortali.
Tutti i Greci sono uomini.
∴ Tutti i Greci sono mortali.

Il primo passo, ovviamente, consiste nel rappresentare in forma simbolica gli enunciati che costituiscono l'argomentazione:

$$\forall x(Hx \rightarrow Mx)$$
$$\forall x(Gx \rightarrow Hx)$$
$$\therefore \forall x(Gx \rightarrow Mx)$$

Usando le due regole per i quantificatori finora introdotte, possiamo fornire la seguente prova della conclusione:

1.	$\forall x(Hx \rightarrow Mx)$	: PREMESSA
2.	$\forall x(Gx \rightarrow Hx)$	: PREMESSA
3.	$(Ha \rightarrow Ma)$	: ∀ELIM,1
4.	$(Ga \rightarrow Ha)$	: ∀ELIM,2
5.	$(Ga \rightarrow Ma)$	: SI,2,1
6.	$\forall x(Gx \rightarrow Mx)$	: ∀INTR,5

A causa delle restrizioni sull'uso di ∀INTR, non è ammessa una derivazione come la seguente:

1.	$\forall x(Gx \rightarrow Mx)$	: PREMESSA
2.	Gf	: PREMESSA
3.	$(Gf \rightarrow Mf)$	: ∀ELIM,1
4.	Mf	: →ELIM,2,3
5.	$\forall xMx$	: ∀INTR,4 [scorretto]

Qui '*f*' compare nella PREMESSA 2; perciò su di esso non si può fondare una generalizzazione.

13.2 REGOLE PER LA QUANTIFICAZIONE ESISTENZIALE

Abbiamo già una coppia di regole per l'introduzione e l'eliminazione di quantificatori universali; ci resta ora da sviluppare una coppia di regole di inferenza per l'introduzione e l'eliminazione dei quantificatori esistenziali.

INTRODUZIONE DEL QUANTIFICATORE ESISTENZIALE

Anche la prossima regola è facile da giustificare. Se qualche cosa è vera per un particolare individuo, allora esiste qualche individuo per cui essa è vera. Schematicamente:

F*a*
∴ ∃*x*F*x*

In modo più formale, questa regola si definisce così:

∃INTR Da un enunciato del tipo
P[c/v]
si può derivare
∃**vP**

In questa regola, la costante **c** sostituisce tutte le occorrenze libere della variabile **v** in una formula ben formata **P**.
La formulazione di questa regola può sembrare strana, perché percorrendo dall'inizio le righe della derivazione, si trova l'enunciato **P[c/v]** prima di raggiungere l'enunciato contenente la variabile **v**, cioè ∃**vP**. Per usare questa regola correttamente, però, ci si deve solo accertare che l'enunciato precedente e la fbf che si vuole quantificare esistenzialmente siano correlate nel modo giusto: il primo si può ottenere da **P** sostituendo con **c** tutte le occorrenze libere di **v**. Inoltre, se si segue una versione modificata di COSTRUZIONE-PROVA, nell'agenda si incontrerà, in effetti, ∃**vP** prima di **P[c/v]**, perché l'agenda comincia dalla fine della derivazione e risale verso le premesse.
Le seguenti applicazioni di ∃INTR sono tutte corrette:

n.	F*aa*	F*aa*	F*aa*	F*aa*	: <PREMESSA o regola>
		.	.	.	
		.	.	.	
		.	.	.	
n + *k*.	∃*x*F*xx*	∃*x*F*xa*	∃*x*F*ax*	∃*x*F*aa*	: ∃INTR,*n*

Tutte queste inferenze sono permesse dalla regola e conservano la verità. Ogni argomentazione avente la riga *n* come premessa e la riga (*n* + *k*) come conclusione è valida. Ciascuna delle quattro semplici inferenze sopra riportate è permessa dalla regola ∃INTR. Inoltre nell'esempio precedente, benché non si possa derivare '∀*x*M*x*' ('Tutto è mortale'), alla riga 5 si può derivare '∃*x*M*x*' ('Qualcosa è mortale'):

1.	∀*x*(G*x*→M*x*)	: PREMESSA
2.	G*f*	: PREMESSA
3.	(G*f*→M*f*)	: ∀ELIM,1
4.	M*f*	: →ELIM,3,2
5.	∃*x*M*x*	: ∃INTR,4

ELIMINAZIONE DEL QUANTIFICATORE ESISTENZIALE

L'ultima regola, ∃ELIM, riguarda le inferenze valide che si possono fare a partire da un enunciato quantificato esistenzialmente. Chiariamo questa regola con un'analogia: spesso, in un'indagine, sappiamo che qualcuno ha commesso il delitto, ma non sappiamo di chi si tratti esattamente. Rappresentiamo questa generica persona con un nome fittizio, ad esempio Mario Rossi, poi ragioniamo su Mario Rossi, benché non sappiamo di chi si tratti in realtà. Ogni conclusione raggiunta a proposito di Mario Rossi, purché non si riferisca a Mario Rossi in persona, è una conclusione corretta.
La strategia che adotteremo nel caso di un enunciato quantificato esistenzialmente consiste nell'indicare con Mario Rossi un individuo non precisato, per vedere che cosa ne segue. Se raggiungiamo una conclusione che non dipende dal fatto che quell'individuo si chiami effettivamente Mario Rossi, allora tale conclusione segue validamente dall'enunciato originario che faceva riferimento a un individuo non specificato. Vediamo la definizione formale della regola e consideriamone alcune applicazioni.

∃ELIM Se un enunciato di una riga precedente è del tipo
∃vP
e se c'è una sottoprova che comincia con l'ASSUNZIONE
P[c/v]
(ove la costante **c** è nuova per la prova) e finisce con un enunciato
Q
che non contiene **c**,
allora **Q** può essere RESTITUITO da tale sottoprova.

Una costante è "nuova per la prova" se non compare precedentemente nella prova. Si noti che la regola RESTITUITO ha così subìto una leggera, ma significativa, estensione.
La regola ∃ELIM si distingue da tutte le altre regole di eliminazione perché non è una regola per eliminare un quantificatore esistenziale da una riga; è piuttosto una strategia per costruire sottoprove che permettano di derivare conclusioni da enunciati quantificati esistenzialmente.
Consideriamo alcuni esempi, tratti ancora dalla tradizionale logica aristotelica.

Tutti gli animali da circo sono addomesticati.
Alcuni leoni sono animali da circo.
∴ Alcuni leoni sono addomesticati.

1. $\forall x(\mathrm{C}x \rightarrow \mathrm{A}x)$: PREMESSA
2. $\exists x(\mathrm{L}x \wedge \mathrm{C}x)$: PREMESSA
/INIZIO: ∃ELIM/
*3. $(\mathrm{L}a \wedge \mathrm{C}a)$: ASSUNZIONE per ∃ELIM,2
*4. $\forall x(\mathrm{C}x \rightarrow \mathrm{A}x)$: INVIATO,1

*5. (C*a*→A*a*) : ∀ELIM,4
*6. C*a* : ∧ELIM,3
*7. A*a* : →ELIM,5,6
*8. L*a* : ∧ELIM,3
*9. (L*a*∧A*a*) : ∧INTR,8,7
*10. ∃*x*(L*x*∧A*x*) : ∃INTR,9
/FINE: ∃ELIM/
11. ∃*x*(L*x*∧A*x*) : RESTITUITO,10

Alla riga 3 abbiamo assunto che *a* sia un leone da circo. La sottoprova si conclude alla riga 10 con un enunciato che non contiene *a* e quindi non dipende dall'assunzione che '*a*' sia il nome di un leone da circo. Perciò l'informazione della riga 10 può essere restituita alla prova principale.

13.3 ALCUNI ESEMPI

Quando si usano le regole di introduzione e di eliminazione dei quantificatori, bisogna accertarsi che il quantificatore introdotto o eliminato abbia per campo di applicazione l'intero enunciato della riga. Esamineremo alcuni modi di trattare enunciati contenenti quantificatori il cui campo è solo una parte dell'enunciato. Ad esempio, in

(∀*x*F*x*∧A)

il campo di '∀*x*' è solo il primo congiunto. La struttura interna del secondo congiunto 'A' qui non interessa: può essere un enunciato qualunque, con un'unica riserva che spiegheremo fra breve. Non possiamo applicare ∀ELIM a questo enunciato così com'è, ma possiamo derivare da esso un altro enunciato a cui ∀ELIM si può applicare:

1. (∀*x*F*x*∧A) : PREMESSA
2. ∀*x*F*x* : ∧ELIM,1
3. A : ∧ELIM,1
4. F*a* : ∀ELIM,2
5. (F*a*∧A) : ∧INTR,4,3
6. ∀*x*(F*x*∧A) : ∀INTR,5

In questa derivazione si suppone che l'enunciato A non contenga la costante '*a*'. Se A contiene delle costanti, la costante introdotta alla riga 4 deve essere diversa da ciascuna di esse.

Vediamo un'altra semplice derivazione che sposta un quantificatore all'inizio di un enunciato:

1. $(A \rightarrow \forall x Fx)$: PREMESSA
/INIZIO: $\rightarrow$INTR per derivare $(A \rightarrow Fa)$/
*2. A : ASSUNZIONE
*3. $(A \rightarrow \forall x Fx)$: INVIATO,1
*4. $\forall x Fx$: $\rightarrow$ELIM,3,2
*5. Fa : $\forall$ELIM,4
*6. $(A \rightarrow Fa)$: $\rightarrow$INTR,2,5
/FINE: $\rightarrow$INTR per descrivere $(A \rightarrow Fa)$/
7. $(A \rightarrow Fa)$: RESTITUITO,6
8. $\forall x(A \rightarrow Fx)$: $\forall$INTR,7

13.4 REGOLA DI NEGAZIONE DI UN QUANTIFICATORE

Prima di passare ai prossimi esempi, converrà considerare i casi in cui un segno di negazione precede un quantificatore il cui campo è il resto dell'enunciato. Ci sono due tipi di casi:

$\neg \forall x Fx$ $\qquad$ $\neg \exists x Fx$

Vogliamo dimostrare che

'$\neg \exists x Fx$' è logicamente equivalente a '$\forall x \neg Fx$'

La dimostrazione dell'equivalenza fra '$\neg \forall x Fx$' e '$\exists x \neg Fx$' è simile e viene lasciata come esercizio al termine del capitolo. Queste equivalenze logiche sono piuttosto importanti, perché permettono di definire delle regole di sostituzione applicabili a fbf contenenti quantificatori accompagnati da segni di negazione. Un modo per dimostrare l'equivalenza si basa su una discussione semantica delle condizioni di verità per la coppia di enunciati. Cominceremo quindi col far notare che

'$\neg \exists x Fx$' vale TRUE se e solo se '$\exists x Fx$' vale FALSE

e che

'$\exists x Fx$' vale FALSE se e solo se ogni esemplare di 'Fx' vale FALSE.

Ma ciò avviene se e solo se ogni esemplare di '$\neg Fx$' vale TRUE, cioè se e solo se '$\forall x \neg Fx$' vale TRUE.
Un secondo modo per dimostrare l'equivalenza consiste nel dimostrare che il bicondizionale dei due termini è un teorema. Dimostreremo cioè '$(\neg \exists x Fx \leftrightarrow \forall x \neg Fx)$' senza partire da alcuna premessa. Durante la derivazione, a

un certo punto si presenterà un problema di strategia; interromperemo allora la derivazione, per discutere il problema e trovare una soluzione.

```
*1.  ¬∃xFx                    : ASSUNZIONE
       /INIZIO: ¬INTR per derivare ¬Fa/
**2.   Fa                     : ASSUNZIONE
**3.   ∃xFx                   : ∃INTR,2
**4.   ¬∃xFx                  : INVIATO,1
**5.   ¬Fa                    : ¬INTR,2,3,4
       /FINE: ¬INTR per derivare ¬Fa/
*6.  ¬Fa                      : RESTITUITO,5
*7.  ∀x¬Fx                    : ∀INTR,6
*8.  (¬∃xFx→∀x¬Fx)            : →INTR,1,7
     /FINE: →INTR/
9.  (¬∃xFx→∀x¬Fx              : RESTITUITO,8
*10.  ∀x¬Fx                   : ASSUNZIONE
       /INIZIO: ¬INTR per derivare ¬∃xFx/
**11.   ∃xFx                  : ASSUNZIONE
          /INIZIO: ∃ELIM/
***12.    Fb                  : ASSUNZIONE per ∃ELIM,2
***13.    ∀x¬Fx               : INVIATO,10
***14.    ¬Fb                 : ∀ELIM,13
```

A questo punto si presenta un problema: alle righe 12 e 14 c'è una contraddizione, ma poiché esse contengono '*b*', cioè la costante dell'assunzione della riga 12, non possono essere restituite al di fuori della sottoprova. Tuttavia, data una contraddizione, si può derivare un enunciato qualsiasi, e in particolare una contraddizione priva della costante '*b*'. Indichiamo con 'A' un enunciato atomico qualsiasi (ad esempio, 'L'erba è verde') e continuiamo la prova.

```
***15.    (Fb∨(A∧¬A))         : ∨INTR,12
***16.    (A∧¬A)              : ∨ELIM,15,14
          /FINE: ∃ELIM/
**17.   (A∧¬A)                : RESTITUITO,16
**18.   A                     : ∧ELIM,17
**19.   ¬A                    : ∧ELIM,17
**20.   ¬∃xFx                 : ¬INTR,11,18,19
       /FINE: ¬INTR per derivare ¬∃xFx/
*21.  ¬∃xFx                   : RESTITUITO,20
*22.  (∀x¬Fx→¬∃xFx)           : →INTR,10,21
      /FINE: →INTR/
23.  (∀x¬Fx→¬∃xFx)            : RESTITUITO,22
24.  (¬∃xFx↔∀x¬Fx)            : ↔INTR,9,23
```

Poiché 'F*x*' non ha svolto nessun ruolo significativo nella prova sopra riportata,

questo risultato vale per qualsiasi fbf che si metta al posto di 'Fx'. Possiamo ora introdurre la seguente *regola di negazione di un quantificatore* (NQ):

NQ: $\neg\exists v$S è derivabile se e solo se è derivabile $\forall v\neg$S
$\neg\forall v$S è derivabile se e solo se è derivabile $\exists v\neg$S.

In qualsiasi punto della prova si può applicare questa regola per far passare il segno di negazione attraverso un quantificatore, purché nel contempo si cambi il tipo del quantificatore. Questa regola derivata è molto utile, come si può vedere nella prova seguente:

Dimostrare che '$\forall x$(F$x\rightarrow$A)' è logicamente equivalente a '($\exists x$F$x\rightarrow$A)'

Dimostriamo il bicondizionale senza usare premesse:

*1. $\forall x$(F$x\rightarrow$A) : ASSUNZIONE
/INIZIO: $\rightarrow$INTR per derivare ($\exists x$F$x\rightarrow$A)/
**2. $\exists x$Fx : ASSUNZIONE
/INIZIO: $\exists$ELIM/
***3. Fa : ASSUNZIONE per $\exists$ELIM,2
***4. $\forall x$(F$x\rightarrow$A) : INVIATO,1
***5. (F$a\rightarrow$A) : $\forall$ELIM,4
***6. A : $\rightarrow$ELIM,3,5
/FINE: $\exists$ELIM/
**7. A : RESTITUITO,6
**8. ($\exists x$F$x\rightarrow$A) : $\rightarrow$INTR,2,7
/FINE: $\rightarrow$INTR per derivare ($\exists x$F$x\rightarrow$A)/
*9. ($\exists x$F$x\rightarrow$A) : RESTITUITO,8
*10. ($\forall x$F$x\rightarrow$A)$\rightarrow$($\exists x$F$x\rightarrow$A)) : INTR,1,9
/FINE: $\rightarrow$INTR/
11. ($\forall x$(F$x\rightarrow$A)$\rightarrow$($\exists x$F$x\rightarrow$A) : RESTITUITO,10
*12. ($\exists x$F$x\rightarrow$A) : ASSUNZIONE
/INIZIO: $\neg$ELIM per derivare $\forall x$(F$x\rightarrow$A)/
**13. $\neg\forall x$(F$x\rightarrow$A) : ASSUNZIONE
**14. $\exists x\neg$(F$x\rightarrow$A) : NQ,13
/INIZIO: $\exists$ELIM/
***15. $\neg$(F$b\rightarrow$A) : ASSUNZIONE per $\exists$ELIM,14
***16. $\neg\neg$(F$b\wedge\neg$A) : RS EQ,15
***17. (F$b\wedge\neg$A) : RS DN,16
***18. Fb : $\wedge$ELIM,17
***19. $\exists x$Fx : $\exists$INTR,18
***20. $\neg$A : $\wedge$ELIM,17
***21. ($\exists x$F$x\wedge\neg$A) : $\wedge$INTR,20,19
/FINE: $\exists$ELIM/
**22. ($\exists x$F$x\wedge\neg$A) : RESTITUITO,21

**23.	$\exists x Fx$	: $\wedge$ELIM,22
**24.	$(\exists x Fx \rightarrow A)$	: INVIATO,12
**25.	A	: $\rightarrow$ELIM,24,23
**26.	$\neg A$	: $\wedge$ELIM,22
**27.	$\forall x(Fx \rightarrow A)$	: $\neg$ELIM,13,25,26
	/FINE: $\neg$ELIM per derivare $\forall x(Fx \rightarrow A)$/	
*28.	$\forall x(Fx \rightarrow A)$	: RESTITUITO,27
*29.	$((\exists x Fx \rightarrow A) \rightarrow \forall x(Fx \rightarrow A))$	: $\rightarrow$INTR,12,28
	/FINE: $\rightarrow$INTR/	
30.	$((\exists x Fx \rightarrow A) \rightarrow \forall x(Fx \rightarrow A))$	: RESTITUITO,29
31.	$(\forall x(Fx \rightarrow A) \leftrightarrow (\exists x Fx \rightarrow A))$	: $\leftrightarrow$INTR,11,30

Applichiamo questo insieme esteso di regole ad alcuni esempi, per acquisire familiarità con le prove. Il primo esempio è di un certo interesse storico. W.S. Jevons, un logico inglese costruttore di una macchina logica, modificando un esempio di Augustus De Morgan, accusò la tradizionale logica aristotelica di non essere capace di dimostrare la validità di questa argomentazione:

I cavalli sono animali.
Perciò ogni testa di cavallo è la testa di un animale.

Indicando 'y è la testa di x' con 'Dyx', otteniamo la seguente rappresentazione simbolica dell'argomentazione:

$\forall x(Cx \rightarrow Ax)$
$\therefore \forall y(\exists x(Cx \wedge Dyx) \rightarrow \exists z(Az \wedge Dyz))$

Lavorando a ritroso a partire dalla conclusione otteniamo la prova seguente:

1.	$\forall x(Cx \rightarrow Ax)$	: PREMESSA
	/INIZIO: $\rightarrow$INTR per derivare $(\exists x(Cx \wedge Dax) \rightarrow \exists z(Az \wedge Daz))$/	
*2.	$\exists x(Cx \wedge Dax)$	: ASSUNZIONE
	/INIZIO: $\exists z(Az \wedge Daz)$/	
**3.	$(Cb \wedge Dab)$	: ASSUNZIONE per $\exists$ELIM,2
**4.	$\forall x(Cx \rightarrow Ax)$	: INVIATO,1
**5.	$(Cb \rightarrow Ab)$	: $\forall$ELIM,4
**6.	Cb	: $\wedge$ELIM,3
**7.	Ab	: $\rightarrow$ELIM,6,5
**8.	Dab	: $\wedge$ELIM,3
**9.	$(Ab \wedge Dab)$	: $\wedge$INTR,7,8
**10.	$\exists z(Az \wedge Daz)$	: $\exists$INTR,9
	/FINE: $\exists$ELIM/	
*11.	$\exists z(Az \wedge Daz)$	: RESTITUITO,10
*12.	$(\exists x(Cx \wedge Dax) \rightarrow \exists z(Az \wedge Daz))$	: $\rightarrow$INTR,2,11
	/FINE: $\rightarrow$INTR/	

13. $(\exists x(Cx \wedge Dax) \rightarrow \exists z(Az \wedge Daz))$: RESTITUITO,12
14. $\forall y(\exists x(Cx \wedge Dyx) \rightarrow \exists z(Az \wedge Dyz))$: $\forall$INTR,13

La prova è così conclusa. Si noti che la riga 13 contiene '*a*', che non viola le condizioni per l'applicazione di ∀INTR, perché le assunzioni delle righe 2 e 3 non sono più in vigore.
Il prossimo esempio illustra il modo in cui si può trattare la relazione di identità con la notazione fin qui introdotta (volendo, comunque, potremmo estendere il sistema attuale in modo da poter trattare separatamente la relazione di identità, con regole di inferenza apposite per formule contenenti un segno di identità ' = ').
Si consideri l'argomentazione seguente:

> Se un evento causa un altro evento, il primo evento comincia prima del secondo. Quando un evento comincia prima di un altro, i due eventi non sono identici. Ogni evento è identico a se stesso. Perciò nessun evento è causa di se stesso.

Poniamo:

Cxy = x causa y
Bxy = x comincia prima di y
Ixy = x è identico a y

La rappresentazione simbolica dell'argomentazione è allora:

1. $\forall x \forall y(Cxy \rightarrow Bxy)$
2. $\forall x \forall y(Bxy \rightarrow \neg Ixy)$
3. $\forall x Ixx$

∴4. $\forall x \neg Cxx$

Si osservi che la conclusione è un enunciato quantificato universalmente; nell'ultimo passaggio della derivazione si dovrà quindi applicare la regola ∀INTR. Come al solito, la strategia consisterà nell'eliminare i quantificatori all'inizio, per poi effettuare alcune trasformazioni di enunciati ed introdurre infine i quantificatori dove servono.

1. $\forall x \forall y(Cxy \rightarrow Bxy)$: PREMESSA
2. $\forall x \forall y(Bxy \rightarrow \neg Ixy)$: PREMESSA
3. $\forall x Ixx$: PREMESSA
4. Iaa : $\forall$ELIM,3
5. $\forall y(Bay \rightarrow \neg Iay)$: $\forall$ELIM,2
6. $(Baa \rightarrow \neg Iaa)$: $\forall$ELIM,5
7. $\forall y(Cay \rightarrow Bay)$: $\forall$ELIM,1
8. $(Caa \rightarrow Baa)$: $\forall$ELIM,7
9. $\neg \neg Iaa$: RS DN,4

10. $\neg Baa$: MT,6,9
11. $\neg Caa$: MT,8,10
12. $\forall x \neg Cxx$: $\forall$INTR,11

L'ultimo esempio illustra un'applicazione della regola di negazione di un quantificatore:

> Non tutte le persone di successo sono ricche. Tutte le persone di successo, però, sono o felici o ricche. Perciò esistono delle persone che non sono ricche ma sono felici.

Rappresentando simbolicamente, con un po' di attenzione, questa argomentazione, otteniamo:

1. $\neg \forall x(Sx \rightarrow Rx)$
2. $\forall x(Sx \rightarrow (Rx \vee Fx))$

$\therefore$3. $\exists x(\neg Rx \wedge Fx)$

Una prova di questa argomentazione è:

1. $\neg \forall x(Sx \rightarrow Rx)$: PREMESSA
2. $\forall x(Sx \rightarrow (Rx \vee Fx))$: PREMESSA
3. $\exists x \neg (Sx \rightarrow Rx)$: NQ,1

/INIZIO: $\exists$ELIM per derivare $\exists x(\neg Rx \wedge Fx)$/

*4. $\neg (Sa \rightarrow Ra)$: ASSUNZIONE per $\exists$ELIM,3
*5. $\forall x(Sx \rightarrow (Rx \vee Fx))$: INVIATO,2
*6. $(Sa \rightarrow (Ra \vee Fa))$: $\forall$ELIM,5
*7. $\neg \neg (Sa \wedge \neg Ra)$: RS EQ,4
*8. $(Sa \wedge \neg Ra)$: RS DN,7
*9. Sa : $\wedge$ELIM,8
*10. $(Ra \vee Fa)$: $\rightarrow$ELIM,9,6
*11. $\neg Ra$: $\wedge$ELIM,8
*12. Fa : $\vee$ELIM,11,10
*13. $(\neg Ra \wedge Fa)$: $\wedge$INTR,11,12
*14. $\exists x(\neg Rx \wedge Fx)$: $\exists$INTR,13

/FINE: $\exists$ELIM/

15. $\exists x(\neg Rx \wedge Fx)$: RESTITUITO,14

La prova è così conclusa. Si consiglia di rivedere questi esempi e di svolgere qualcuno degli esercizi riportati alla fine di questo capitolo.

13.5 ARGOMENTAZIONI NON VALIDE

Abbiamo derivato conclusioni di argomentazioni valide. Che cosa succede invece se un'argomentazione non è valida? Come si può dimostrare che un'argomentazione non è valida? Si consideri questa argomentazione:

Tutti gli animali da circo sono addomesticati.
Alcuni leoni non sono animali da circo.
∴ Alcuni leoni non sono addomesticati.

$\forall x(Cx \rightarrow Ax)$
$\exists x(Lx \wedge \neg Cx)$
$\therefore \exists x(Lx \wedge \neg Ax)$

Per quanto si possa tentare, non si riuscirà mai a derivare la conclusione indicata usando le regole di inferenza ammesse.
Questa impossibilità è dovuta al fatto che la conclusione non segue validamente dalle premesse; ma come lo si può dimostrare? Per dimostrare che un'argomentazione non è valida dobbiamo trovare un modello in cui, dall'analisi dei dossier degli individui, si possa rilevare che le premesse valgono TRUE, mentre la conclusione vale FALSE. Per l'argomentazione che stiamo considerando esistono molti modelli di questo tipo; eccone uno con due soli individui:

	C	A	L
a	1	1	0
b	0	1	1

Come si può vedere, in questo modello valgono TRUE sia '$(Ca \rightarrow Aa)$' sia $(Cb \rightarrow Ab)$'. Perciò in questo modello

'$\forall x(Cx \rightarrow Ax)$' vale TRUE.

Inoltre $V(Lb \wedge \neg Cb) =$ TRUE; dunque

'$\exists x(Lx \wedge \neg Cx)$' vale TRUE nel modello.

Invece $V(La \wedge \neg Aa) =$ FALSE e $V(Lb \wedge \neg Ab) =$ FALSE. Poiché non vi sono altri individui,

'$\exists x(Lx \wedge \neg Ax)$' vale FALSE nel modello.

Non esiste nessun algoritmo che permetta di trovare dei modelli che invalidino un'argomentazione. È però possibile definire dei procedimenti e delle regole em-

piriche per la soluzione di questo problema di ricerca, come vedremo nel prossimo capitolo.

13.6 CONCLUSIONI

Abbiamo definito due regole per la quantificazione universale: l'eliminazione del quantificatore universale (∀ELIM) e l'introduzione del quantificatore universale (∀INTR).

∀ELIM - Da un enunciato del tipo **∀vP** si può derivare **P[c/v]**.
∀INTR - Da un enunciato **P** si può derivare **∀vP[v/c]**, purché:

1. **c** non compaia in nessuna premessa;
2. se **P** è in una sottoprova, nessuna costante di **P** compaia in un'ASSUNZIONE tuttora in vigore;
3. tutte le nuove occorrenze di **v** in **P** siano libere dopo la sostituzione **P[v/c]**.

La notazione **P[c/v]** significa che la costante **c** sostituisce tutte le occorrenze libere della variabile **v** in **P**. Analogamente, **P[v/c]** significa che la variabile **v** sostituisce tutte le occorrenze della costante **c** in **P** e che le occorrenze di **v** sono libere dopo la sostituzione.
Abbiamo anche definito due regole per la quantificazione esistenziale: l'introduzione del quantificatore esistenziale (∃INTR) e l'eliminazione del quantificatore esistenziale (∃ELIM).

∃INTR - Da un enunciato **P[c/v]** si può derivare **∃vP**.
∃ELIM - Se un enunciato è del tipo **∃vP** e se c'è una sottoprova avente come ASSUNZIONE **P[c/v]**, dove **c** è una costante che non compare precedentemente nella prova, e se tale sottoprova termina con un enunciato **Q** che non contiene **c**, allora **Q** può essere RESTITUITO dalla sottoprova.

Abbiamo anche dimostrato (in parte) la regola di negazione di un quantificatore:

¬**∃vS** è derivabile se e solo se è derivabile **∀v**¬**S**.
¬**∀vS** è derivabile se e solo se è derivabile **∃v**¬**S**.

Questa regola permette di spostare un segno di negazione attraverso un quantificatore, cambiando nel contempo il tipo del quantificatore.
Dopo aver considerato alcuni esempi, abbiamo infine accennato al problema della dimostrazione della non validità di un'argomentazione.

13.7 ESERCIZI

A. Costruire una derivazione per ognuna delle argomentazioni seguenti:

1. $\forall x(Fx \to Gx)$
   $\exists x(Fx \land Hx)$
   $\therefore \exists x(Gx \land Hx)$
2. $\neg \forall x(Fx \to Gx)$
   $\therefore \exists x(Fx \land \neg Gx)$
3. $\neg \exists x(Fx \land \neg Gx)$
   $\therefore \forall x(Fx \to Gx)$
4. $\forall x(Fx \to \exists y Rxy)$
   $\forall x \forall y(Rxy \to Gx)$
   $\exists x Fx$
   $\therefore \exists x Gx$
5. $\forall x(Fx \to Gx)$
   $(\exists x Gx \to \exists x(Hx \land Dx))$
   $\therefore (\exists x Fx \to \exists x Hx)$
6. $\neg \exists x Fx$
   $\therefore \forall x(Fx \to Gx)$
7. $\neg \forall x(Fx \to \neg Gx)$
   $\therefore \exists x(Fx \land Gx)$
8. $\neg \exists x(Fx \land Gx)$
   $\therefore \forall x(Fx \to \neg Gx)$
9. $\forall x \neg Gx$
   $\forall x \forall y(Rxy \to Fx)$
   $\forall x(Fx \to Gx)$
   $\therefore \exists x \exists y \neg Rxy$
10. $\forall x((Fx \lor Gx) \to Hx)$
    $\forall x((Hx \lor Dx) \to \neg Fx)$
    $\therefore \forall x \neg Fx$

B. Provare l'equivalenza logica delle seguenti coppie di enunciati, dimostrando che il loro bicondizionale è un teorema.

1.	$\forall x(Fx \land Gx)$	$(\forall x Fx \land \forall x Gx)$
2.	$\forall x(Fx \land A)$	$(\forall x Fx \land A)$
3.	$\exists x(Fx \lor Gx)$	$(\exists x Fx \lor \exists x Gx)$
4.	$\forall x(Fx \lor A)$	$(\forall x Fx \lor A)$
5.	$\forall x(A \to Fx)$	$(A \to \forall x Fx)$
6.	$\exists x(Fx \to A)$	$(\forall x Fx \to A)$
7.	$\forall x \forall y Fxy$	$\forall y \forall x Fxy$
8.	$\exists x \exists y Fxy$	$\exists y \exists x Fxy$
9.	$\neg \forall x \exists y Fxy$	$\exists x \forall y \neg Fxy$
10.	$\forall x((Fx \lor Gx) \to Hx)$	$\forall x((Fx \to Hx) \land (Gx \to Hx))$

C. I *teoremi logici* sono enunciati derivabili senza partire da alcuna premessa; se le regole di inferenza sono scelte correttamente, si tratta di enunciati universalmente validi. Dimostriamo un teorema logico:

Dimostrare $(\exists y \forall x Fxy \to \forall x \exists y Fxy)$

/INIZIO: →INTR per derivare la conclusione/

*1.	$\exists y \forall x Fxy$	: ASSUNZIONE
	/INIZIO: ∃ELIM/	
**2.	$\forall x Fxa$	: ASSUNZIONE per ∃ELIM,1
**3.	Fba	: ∀ELIM,2

**4.	$\exists y Fby$	: ∃INTR,3
	/FINE: ∃ELIM/	
*5.	$\exists y Fby$	: RESTITUITO,4
*6.	$\forall x \exists y Fxy$	: ∀INTR,5
*7.	$(\exists y \forall x Fxy \rightarrow \forall x \exists y Fxy)$	: →INTR,1,6
	/FINE: →INTR/	
8.	$(\exists y \forall x Fxy \rightarrow \forall x \exists y Fxy)$	: RESTITUITO,7

Si noti che, nella prova di un teorema logico l'ultima riga è priva di asterischi. Dimostrare i seguenti teoremi logici:

1. $\exists x(Fx \rightarrow \forall x Fx)$
2. $((\exists x Fx \rightarrow \forall x Fx) \rightarrow (\forall x Fx \vee \forall x \neg Fx))$
3. $(\forall x(Fx \rightarrow Gx) \rightarrow (\exists x \neg Gx \rightarrow \exists x \neg Fx))$
4. $(\exists x(\exists y Fy \rightarrow Gx(\exists x Fx \rightarrow \exists y Gy))$
5. $\neg \exists y \forall x(Fxy \leftrightarrow \neg Fxx)$

D. Rappresentare simbolicamente ciascuna delle argomentazioni seguenti e fornire poi una prova.
   1. Tutti i fenomenalisti negano la realtà della materia, mentre nessun materialista la nega. Perciò nessun materialista è un fenomenalista.
   2. Nessun capitalista è socialista. Solo i socialisti sono egualitari. Perciò nessun capitalista è egualitario.
   3. Tutti i personaggi politici sono dotati di comunicativa. Alcune donne sono personaggi politici. Perciò alcune donne sono dotate di comunicativa.
   4. Tutti gli studenti scelgono o logica o matematica. Alcuni studenti non scelgono matematica. Perciò alcuni studenti scelgono logica.
   5. Chiunque aiuti un criminale è colpevole. Perciò ogni criminale che aiuti se stesso è colpevole.
   6. Se Aldo si laurea, allora si laureano tutti. Aldo si laurea solo se si laurea anche Elisa. Ma Elisa si laurea solo se si laureano tutti. Perciò, se qualcuno non si laurea, non si laureano né Aldo né Elisa.
   7. Nessuno che pensi ai propri interessi appoggia tutte le decisioni del partito. Una persona è completamente leale se appoggia tutte le decisioni del partito. Perciò chi è completamente leale non pensa ai propri interessi.
   8. Alcuni insegnanti sono ammirati da tutti gli studenti che ammirano qualche insegnante. Ogni studente ammira qualche insegnante. Perciò ci sono degli insegnanti che sono ammirati da tutti gli studenti.
   9. Ognuno apprezza ogni cosa che sia apprezzata da qualcuno che egli apprezza. Non tutti disprezzano tutti. Ognuno apprezza chi lo apprezza. Perciò qualcuno apprezza se stesso.

10. Ogni volta che la scuola ha un problema, il consiglio di istituto ne attribuisce la colpa al preside. Se qualcuno attribuisce a una persona la colpa di qualcosa, vuol dire che pensa che quella persona sia responsabile di quella cosa per cui è incolpata. Il preside è una persona. Perciò c'è una persona che il consiglio d'istituto ritiene responsabile di tutti i problemi della scuola.

14

Logica predicativa: determinazione della validità di un'argomentazione e dimostrazione automatica di teoremi

Come abbiamo detto nel Capitolo 2, l'Intelligenza Artificiale studia come programmare i calcolatori per far loro eseguire delle attività che richiedono capacità di ragionamento. Fra queste attività rientrano la determinazione della validità di un'argomentazione, la deduzione di conseguenze valide a partire da enunciati dati e la costruzione di prove di argomentazioni valide.

Nel Capitolo 6 abbiamo definito un algoritmo per determinare se un'argomentazione in logica enunciativa è valida. Abbiamo visto che l'applicazione di questo metodo è del tutto automatica e permette di determinare in un tempo finito se un'argomentazione qualsiasi è valida oppure no; perciò questo metodo è un algoritmo. Nel Capitolo 11 abbiamo definito un metodo per costruire una prova di un'argomentazione valida in logica enunciativa. Benché l'applicazione di questo metodo sia automatica, o possa facilmente essere resa tale, non sempre permetterà di produrre una prova corretta di un'argomentazione valida in un tempo finito. Perciò questo metodo non è un vero algoritmo. Si può però dimostrare che un algoritmo per costruire prova in logica enunciativa può essere definito, anche se risulta più lungo e meno comodo da usare rispetto al metodo che abbiamo definito noi.

Nei Capitolo 12 e 13 abbiamo introdotto i concetti della logica predicativa, che vanno ben al di là di quelli usati in logica enunciativa. Mentre la logica enunciativa tratta le argomentazioni senza esaminare la struttura interna degli enunciati atomici, la logica predicativa esamina anche il modo in cui sono costruiti gli enunciati atomici.

14.1 DECIDIBILITÀ

Per ogni sistema logico si possono porre due domande importanti:

1. È possibile definire un algoritmo per determinare la validità di qualsiasi argomentazione nel sistema logico?
2. Esiste un procedimento automatico che produca sempre una prova di qualsiasi argomentazione valida nel sistema logico?

Il primo problema riguarda la *decidibilità* del sistema logico: ci si chiede se esista un algoritmo in grado di decidere se un'argomentazione è valida oppure no. Non ci siamo posti esplicitamente questo problema a proposito della logica enunciativa, perché, una volta introdotte le tavole di verità, come si è fatto nel Capitolo 6, è facile rendersi conto che la logica enunciativa è decidibile.
La logica predicativa, invece, anche se ciò può sembrare sorprendente, non è decidibile: non esiste nessun algoritmo che possa classificare ogni argomentazione come valida o non valida. È essenziale rendersi conto della portata di questa affermazione: non diciamo semplicemente che un tale algoritmo è difficile da trovare o non è ancora stato trovato, né che è lungo e difficile da descrivere; diciamo invece che è impossibile ideare un tale algoritmo. Questo risultato di grande rilevanza è detto *teorema di Church*, dal nome del logico statunitense Alonzo Church, che lo dimostrò nel 1936.
Il teorema di Church implica che un capitolo come il 6 non è possibile per la logica predicativa: non esiste niente di simile a una tavola di verità che si possa generare con un algoritmo e applicare a un'argomentazione in logica predicativa per determinare se essa è valida o no. La spiegazione del ragionamento irrefutabile su cui si fonda il teorema di Church va al di là degli scopi di questo libro. In un enunciato, le difficoltà identificate dal teorema di Church sorgono quando si ammettono sia i quantificatori, sia le relazioni.

14.2 MODELLI

Esistono alcuni tipi di argomentazioni che sfuggono al teorema di Church e per i quali è quindi possibile costruire un algoritmo che classifichi le argomentazioni come valide o non valide; come abbiamo visto, è questo il caso della logica enunciativa.
Chiamiamo *logica predicativa non ristretta* la logica in cui le formule possono contenere quantificatori, relazioni, connettivi proposizionali e gli altri simboli logici che abbiamo trattato. Il teorema di Church afferma che la logica predicativa non ristretta è indecidibile.
In logica predicativa esistono dei tipi di argomentazioni che sono decidibili, benché non esista una procedura di decisione applicabile a qualsiasi argomentazione? Come ci si potrebbe aspettare, la risposta a questa domanda è affermativa, ma

una discussione approfondita e una dimostrazione di questo fatto vanno al di là degli scopi di questo libro.
Il problema della validità di una tale argomentazione equivale al problema seguente:

> Esiste un modello in cui le premesse siano soddisfatte (valgano TRUE) ma la conclusione non sia soddisfatta (valga FALSE)?

Se ciò avviene, l'argomentazione non è valida; altrimenti è valida. Questo fatto può essere espresso sotto forma di algoritmo:

1. INPUT un'argomentazione.
2. FOR ogni possibile modello per questa argomentazione
   (a) IF nel modello le premesse sono soddisfatte e la conclusione non è soddisfatta
   THEN OUTPUT "Non valida" e STOP.
3. OUTPUT "Valida" e STOP.

Per poter usare questo algoritmo dobbiamo raffinare il passo 2. In particolare, abbiamo bisogno di un procedimento per costruire tutti i modelli possibili per un'argomentazione e di un criterio di ordinamento dei modelli che ci permetta di esaminarli tutti con il ciclo FOR. Dobbiamo anche essere sicuri che esista solo un numero finito di modelli, in modo da esser certi che prima o poi si uscirà dal ciclo FOR, o perché sono stati esaminati tutti i modelli senza trovarne uno in cui tutte le premesse siano soddisfatte ma la conclusione non lo sia, nel qual caso si va al passo 3, o perché è stato trovato un modello in cui questa condizione si verifica, nel qual caso si esce prematuramente dal ciclo, eseguendo l'azione indicata al passo 2(a).
Le argomentazioni prive di quantificatori sono decidibili: basta trattare 'F*a*', '*Gb*' o 'H*bc*' come enunciati atomici, ciascuno dei quali vale o TRUE o FALSE in un dato modello. In questo caso il numero dei modelli è ovviamente finito. Si può inoltre applicare direttamente l'ALGORITMO DI WANG: secondo la terminologia introdotta al Capitolo 6, se uno stesso enunciato compare sia nella lista destra, sia in quella sinistra, il tentativo fallisce; se tutti i tentativi falliscono, l'argomentazione è valida.
Anche le argomentazioni contenenti solo predicati a un argomento (oltre ad eventuali enunciati atomici) sono decidibili. Se nell'argomentazione ci sono *n* predicati distinti a un argomento, allora per determinare se l'argomentazione è valida basta considerare al più 2**n tipi distinti di individui. Ogni volta che si deve trattare con un numero finito di individui è possibile costruire un algoritmo di decisione. In questo caso, ogni enunciato quantificato universalmente si riduce a una congiunzione finita di esemplari e ogni enunciato quantificato esistenzialmente si riduce a una disgiunzione finita di esemplari; perciò si possono usare le tavole di verità o l'ALGORITMO DI WANG.
I problemi si presentano quando si hanno relazioni quantificate a *n* argomenti,

con $n > 1$, perché aumenta indefinitamente il numero dei tipi di individui distinti, e quindi il numero dei modelli possibili. Si consideri

$\exists x Fxa$

Cominciamo da un modello con un solo tipo di individuo, ad esempio Alfredo: la lettera '*a*' si riferisce ad Alfredo. Ora, 'F*aa*' può valere TRUE, nel qual caso vale TRUE anche '∃*x*F*xa*'. Per esaminare tutte le situazioni possibili ci servono perciò due tipi di individui: gli individui per cui 'F*aa*' vale TRUE e quelli per cui 'F*aa*' vale FALSE. Alfredo sia un individuo del primo tipo e Beatrice del secondo: quando '*a*' indica Alfredo si ha V(F*aa*) = TRUE, mentre quando '*a*' indica Beatrice si ha V(F*aa*) = FALSE. A questo punto ci serve un modello che contenga sia Alfredo sia Beatrice, che indicheremo con '*a*' e '*b*' rispettivamente (ovviamente potremmo anche scambiare fra loro i nomi, oppure ammettere che entrambi si riferiscano allo stesso individuo, ad esempio Alfredo). Ma quali sono i valori di verità di 'F*ab*' e di 'F*ba*'? Il numero dei tipi possibili di individui raddoppia ogni volta. Se continuiamo, il numero dei tipi possibili di individui tende rapidamente all'infinito, e il numero dei modelli vi tende ancora più rapidamente, perché i modelli sono sottoinsiemi dell'insieme degli individui.
Da quando è stato dimostrato il teorema di Church, gli studiosi di logica hanno cercato di identificare non solo i tipi di argomentazioni decidibili, ma anche quelli indecidibili per qualsiasi algoritmo. Alcuni studiosi di logica hanno anzi dimostrato che alcuni tipi di argomentazione sono *più indecidibili* di altri. Il curioso concetto di maggiore indecidibilità può essere definito in modo rigoroso. Il tipo di argomentazioni A si dice più indecidibile del tipo di argomentazioni B se, dato un modo per decidere le argomentazioni di tipo A (ad esempio un'ispirazione divina), si possono decidere le argomentazioni di tipo B, mentre dato un modo (un'altra ispirazione divina) per decidere le argomentazioni di tipo B, rimane impossibile decidere le argomentazioni di tipo A. Purtroppo una trattazione più approfondita di questo argomento va ben al di là degli scopi di questo libro.

14.3 TESI DI CHURCH

Può esistere un calcolatore che, per qualsiasi argomentazione, sia in grado di decidere se è valida o no? Ciò che abbiamo detto finora non implica direttamente che un tale calcolatore non possa esistere; abbiamo solo affermato che:

> Non può esistere nessun algoritmo che classifichi le argomentazioni come valide o non valide.

Per poter affermare che nessun calcolatore potrà mai effettuare tale classificazione abbiamo bisogno di un'ulteriore assunzione:

Un calcolatore è in grado di eseguire solo ciò che è descritto da un algoritmo.

Quest'ultima affermazione è largamente accettata come vera, ma non può essere dimostrata in modo definitivo; essa è nota come *tesi di Church*, o *tesi di Church-Turing*. Non deve essere confusa con il teorema di Church, che è stato definitivamente dimostrato.
Abbiamo così la seguente argomentazione valida riguardante i limiti dei calcolatori:

1. Non può esistere nessun algoritmo che classifichi le argomentazioni in valide o non valide (teorema di Church).
2. Ogni calcolatore è in grado di eseguire solo ciò che è descritto da un algoritmo (tesi di Church).

∴ 3. Nessun calcolatore potrà mai classificare in modo effettivo tutte le argomentazioni come valide o non valide.

La validità della tesi di Church è discussa nell'Appendice B.
Qualcuno ha sfruttato questa debolezza dei calcolatori, consistente nell'incapacità di classificare in modo effettivo tutte le argomentazioni come valide o non valide, per affermare che ciò dimostra che esistono attività eseguibili da un essere umano, ma non da un calcolatore. È vero che il teorema di Church, insieme alla tesi di Church, impone dei limiti alla capacità di ragionare dei calcolatori. Sembra però che anche gli esseri umani siano soggetti agli stessi limiti: esistono degli esseri umani che siano in grado di stabilire sempre, correttamente e in un tempo finito, se un'argomentazione è valida o meno? Sembra di no; in effetti, ben pochi di noi sono capaci di determinare se un'argomentazione relativamente complicata, in logica predicativa, è valida o no, e comunque ciò richiede un bel po' di tempo. Non diciamo per questo che tali limiti, cui sono soggetti gli esseri umani, comportano automaticamente che noi siamo incapaci di ragionare; non dovremmo dunque affermare ciò neanche a proposito dei calcolatori. Perciò il fatto che i calcolatori non siano in grado di decidere, per ogni argomentazione, se è valida o no, non dimostra in modo decisivo la superiorità del ragionamento umano, in quanto è evidente che siamo soggetti alle stesse limitazioni.

14.4 DIMOSTRAZIONE AUTOMATICA DI TEOREMI

Il teorema di Church e la tesi di Church sono un duro colpo, perché dimostrano chiaramente i limiti dell'intelligenza nel risolvere problemi, indipendentemente dal fatto che questa intelligenza sia artificiale o umana. Alla luce di queste osservazioni, potremmo affrontare con un certo pessimismo la seconda domanda che ci eravamo posti all'inizio:

Se è noto in qualche modo che un'argomentazione in logica predicativa è

valida, esiste un procedimento automatico che sia sempre in grado di produrne una prova?

Dopo lo scossone che il nostro senso di sicurezza ha subito a causa del teorema di Church, potremmo essere tentati di rispondere "Probabilmente no!". Contrariamente a quanto ci si potrebbe aspettare, tuttavia, la risposta a questa domanda è affermativa: tale procedimento esiste. Se un'argomentazione è valida, esiste un metodo che finirà per dimostrarne la validità; inoltre tale metodo può essere concepito in modo che fornisca sempre una dimostrazione che mostri *perché* l'argomentazione è valida. È interessante considerare che cosa succederebbe applicando questo procedimento a un'argomentazione non valida; in questo caso entra in gioco il teorema di Church: il metodo non giungerà mai a termine. Se un calcolatore fosse programmato per applicare questo metodo e ricevesse in ingresso un'argomentazione non valida, non sapremmo mai se il calcolatore sta per fornirci una prova o una confutazione, oppure se è destinato a lavorare per sempre sul problema.
La questione che stiamo affrontando rientra nell'ambito più generale della *dimostrazione automatica di teoremi.* Ci sono essenzialmente due modi diversi di costruire un procedimento per dimostrare la validità di un'argomentazione. Un modo è simile al metodo COSTRUZIONE-PROVA visto nel Capitolo 11, ove avevamo definito un procedimento quasi automatico che forniva in uscita una prova nel sistema deduttivo che avevamo adottato. Questo metodo dimostra la validità di un'argomentazione fornendone una prova.
L'altro modo di creare un procedimento per dimostrare la validità di un'argomentazione non fornisce in uscita una prova vera e propria: anziché fornire una prova basata sulle regole di deduzione dimostra che è impossibile che le premesse valgano TRUE e la conclusione valga FALSE.
Si consideri l'argomentazione seguente:

A
B
∴C

Come abbiamo visto nel Capitolo 7, questa argomentazione è valida se e solo se il condizionale ad essa corrispondente

$((A \wedge B) \rightarrow C)$

è una tautologia. In altre parole, l'argomentazione è valida se e solo se questo condizionale vale TRUE in ogni possibile situazione. Equivalentemente, l'argomentazione è valida se e solo se la congiunzione

$((A \wedge B) \wedge \neg C)$

è contraddittoria, cioè se e solo se è possibile derivarne una contraddizione. In altre parole, l'argomentazione è valida se e solo se questa congiunzione vale FALSE in ogni possibile situazione (per riprendere i concetti di 'tautologia', 'enunciato contraddittorio' e 'contraddizione', si rileggano i relativi paragrafi del Capitolo 7).

RISOLUZIONE

Esiste una tecnica che si fonda su questa seconda considerazione: un'argomentazione è valida se e solo se la congiunzione di tutte le premesse con la negazione della conclusione non può essere soddisfatta in nessun modello. Questa tecnica è detta *risoluzione*. A prima vista può non sembrare molto promettente questa affermazione secondo la quale un'argomentazione è valida se e solo se una certa formula vale FALSE in ogni situazione possibile: dopo tutto, per determinare se una formula vale FALSE in ogni situazione potrebbe occorrere un tempo infinito. Vediamo come funziona la risoluzione nel caso della logica enunciativa. La risoluzione si basa sul fatto che molti tipi di argomentazioni valide, compresa la maggior parte delle regole di inferenza, possono essere considerati come esemplari di un solo modello molto generale, che può essere usato come unica regola di inferenza: la *regola di risoluzione*. Per rendercene conto meglio, consideriamo una nuova notazione per la logica enunciativa. Conveniamo di rappresentare una disgiunzione

$$(\mathbf{P} \lor \mathbf{Q})$$

con

$$\mathbf{PQ}$$

e una negazione

$$\neg \mathbf{P}$$

con

$$\mathbf{P}'$$

Conveniamo poi di scrivere le premesse di un'argomentazione al di sopra di una linea orizzontale, separandole con virgole, e di riportare la conclusione sotto tale linea. Consideriamo ora alcune forme valide di argomentazione, convertendo le premesse e la conclusione prima in FNC e poi in questa nuova notazione:

$(P \rightarrow Q)$
P
$\therefore Q$

diventa $$\frac{P'Q,\ P}{Q}$$

$(P \rightarrow Q)$
$\neg Q$
$\therefore \neg P$

diventa $$\frac{P'Q,\ Q'}{P'}$$

$((P \wedge (Q \wedge R)) \rightarrow S)$
Q
$\therefore ((P \wedge R) \rightarrow S)$

diventa $$\frac{P'Q'R'S,\ Q}{P'R'S}$$

$(P \rightarrow Q$
$(Q \rightarrow R)$
$\therefore (P \rightarrow R)$

diventa $$\frac{P'Q,\ Q'R}{P \rightarrow R}$$

$(P \vee Q)$
$\neg P$
$\therefore Q$

diventa $$\frac{PQ,\ P'}{Q}$$

Considerando queste argomentazioni valide, espresse nella nuova notazione, ci si accorge che tutte hanno delle caratteristiche in comune:

1. Ogni premessa è una disgiunzione di enunciati.
2. Una premessa contiene un disgiunto **U**.
3. L'altra premessa contiene un disgiunto **U**$'$.
4. La conclusione è la disgiunzione di tutti gli enunciati delle premesse, eccettuati **U** e **U**$'$.

In generale, tutte le argomentazioni del tipo

$$\frac{U'QR\ldots,\ UST\ldots}{QRST\ldots}$$

sono valide. Questa è la regola di risoluzione.
Il seguente algoritmo usa la risoluzione per decidere se un'argomentazione in logica enunciativa è valida; esso comincia col convertire in forma normale congiuntiva (FNC) gli enunciati dell'argomentazione. La FNC e un relativo algoritmo di conversione sono stati definiti nel Capitolo 7.

1. INPUT un'argomentazione.
2. LET C = insieme costituito dalle premesse e dalla negazione della conclusione dell'argomentazione.

3. FOR ogni enunciato S dell'insieme C
   (a) Sostituire S con il suo equivalente in FNC.
4. Ripetere i passi seguenti:
   (a) Trovare nell'insieme C due enunciati S_1 e S_2 tali che S_1 contenga come disgiunto un enunciato atomico **P** e S_2 contenga come disgiunto la sua negazione ¬**P**.
   (b) LET S = nuovo enunciato in FNC, i cui disgiunti sono tutti i disgiunti di S_1 e di S_2, eccettuati **P** e ¬**P**. (S è detto *risolvente* di S_1 e S_2)
   (c) Eliminare S_1 e S_2 da C.
   (d) Aggiungere S a C.

   finché (i) C contenga due enunciati **Q** e ¬**Q**
   oppure
   (ii) non si possa formare nessun nuovo risolvente S.
5. IF si è usciti dal ciclo a ripetizione del passo 4 perché si è verificata la condizione (i)
   THEN OUTPUT "Valida" e STOP.
6. IF si è usciti dal ciclo a ripetizione del passo 4 perché si è verificata la condizione (ii)
   THEN OUTPUT "Non valida" e STOP.

Si ricordi che stiamo cercando di derivare una contraddizione dall'insieme C. Il passo 4 è un ciclo che cerca ripetutamente una contraddizione generando e verificando risolventi successivi. Ogni risolvente è logicamente derivabile dai due enunciati usati per generarlo; perciò, se si produce una contraddizione, si può uscire dal ciclo concludendo che l'argomentazione è valida. Vediamo un paio di esempi che illustrino questo metodo.

Esempio 1.

1. (A→B)
   A
   ∴B
2. C = {(A→B), A, ¬B}
3. C = {(¬A∨B), A, ¬B}
4. Si prende S_1 = (¬A∨B) e S_2 = A.
   Allora S = B.
   Perciò C = {B, ¬B}.

Si è verificata la condizione (i), quindi l'argomentazione è valida (come si può verificare con una semplice applicazione di →ELIM).

Esempio 2.

1. $\neg(\neg A \wedge \neg B)$
   B
   $\therefore A$
2. $C = \{\neg(\neg A \wedge \neg B),\ B,\ \neg A\}$
3. $C = \{(A \vee B), B, \neg A\}$

Si è verificata la condizione (ii), quindi l'argomentazione non è valida.

Per le argomentazioni in logica predicativa è necessario trovare degli esemplari di formule quantificate e applicare ad essi la tecnica sopra descritta. Il problema fondamentale consiste nel trovare gli esemplari appropriati. Nel 1930 il logico francese Jacques Herbrand dimostrò che se una formula vale FALSE in un certo numero finito di situazioni (descritte nel cosiddetto *universo di Herbrand*), allora vale FALSE in tutte le situazioni. Perciò, in virtù del teorema di Herbrand, in effetti non dobbiamo considerare tutte le situazioni, o tutti gli esemplari, per vedere se una formula è contraddittoria: ci basta verificare se vale FALSE in un certo numero finito di situazioni. Se essa vale FALSE in tutte queste situazioni, possiamo concludere che vale FALSE in tutte le situazioni possibili, cioè è contraddittoria. Herbrand dimostrò che il suo teorema era, per così dire, valido in astratto: non indicò come lo si potesse effettivamente applicare in modo conveniente a una formula, bensì si limitò a dimostrare che esiste un procedimento automatico per dimostrare che un'argomentazione non è valida.
Nel 1930 non esisteva un calcolatore che si potesse usare per sfruttare le conseguenze del teorema di Herbrand. Negli anni Sessanta, invece, fu riconosciuta l'importanza del teorema di Herbrand per la dimostrazione automatica di teoremi, e diversi logici e informatici cercarono un modo di applicarlo col calcolatore. Purtroppo i primi tentativi non riuscirono completamente: per alcune formule non si riusciva a dimostrare che fossero contraddittorie (anche quando lo erano veramente), mentre per altre tale dimostrazione richiedeva un tempo di calcolo lunghissimo.
Notevoli progressi furono fatti dai logici e dagli informatici che svilupparono la risoluzione nei primi anni Sessanta. Come abbiamo visto, questa tecnica è un metodo per dimostrare come una formula sia contraddittoria: essa evidenzia una contraddizione nascosta da qualche parte nella formula. La tecnica di risoluzione si rivelò una vera procedura automatica, nel senso che, se una formula è contraddittoria, riesce sempre a scoprirlo in un tempo finito. La tecnica di risoluzione permise anche di scrivere programmi per calcolatore in grado di scoprire queste contraddizioni in un tempo ragionevolmente breve.
Anche il metodo di risoluzione per la dimostrazione di teoremi è soggetto alle limitazioni imposte dal teorema di Church: se viene applicato a un'argomentazione non valida, non giunge mai a termine. Un calcolatore che sia programmato per applicare questo metodo lavorerà su un tale problema finché non lo interrompia-

mo dall'esterno, dopo un giorno o dopo anni. Una volta che lo abbiamo interrotto, non potremo sapere se l'argomentazione non era valida (per cui il programma non sarebbe mai giunto a termine), oppure se era valida, ma era uno di quei problemi la cui soluzione richiede giorni o anni. Per ulteriori informazioni sulla risoluzione, vedi Raphael, 1976.

IL METODO DELLA PROVA

Ritorniamo sul metodo di dimostrazione automatica di teoremi basato sulla costruzione di una prova, perché di solito è più informativo. Ci chiediamo se sia possibile modificare COSTRUZIONE-PROVA in modo da tener conto anche delle nuove regole riguardanti i quantificatori, le costanti, i predicati e le variabili. Il metodo di costruzione di una prova così modificato sarà chiamato COSTRUZIONE-PROVA*, per distinguerlo dal suo predecessore, applicabile solo in logica enunciativa.
Per definire le modifiche necessarie partiamo da alcune considerazioni sulla strategia di utilizzazione dei quantificatori e delle costanti. Supponiamo di avere tre premesse 'A', 'B' e 'C', la cui struttura interna ora non ci interessa, e una conclusione

$\forall x Fx$

Come possiamo procedere per derivare questa conclusione? Potremmo ragionare nel modo seguente: se in qualche modo abbiamo già derivato 'F*b*', possiamo inferirne '$\forall x Fx$', purché 'F*b*', sia stato ottenuto nel modo opportuno. Una volta derivato 'F*b*', si applica ∀INTR, ottenendo $\forall x Fx$, come si voleva. Pensiamo invece quale strategia dovremmo adottare se la premessa 'C' avesse la struttura

$(B \rightarrow \forall x Fx)$

In questo caso non sembra necessario derivare 'F*b*', perché '$\forall x Fx$' si può derivare direttamente, se si riesce a derivare 'B'. Queste due osservazioni costituiscono dei suggerimenti per una parte delle modifiche. Le due possibili strategie che abbiamo appena esposto in realtà sono in parte già contemplate dai passi 3(a) e 3(b) di COSTRUZIONE-PROVA:

3(a) IF **OB** è una sottoformula di un enunciato **Q** di una riga accessibile ...
3(b) IF **OB** non è una sottoformula di un enunciato di una riga accessibile...

Questi passi servono per decidere se l'obiettivo debba essere derivato da una riga precedente oppure debba essere costruito mediante qualche strategia predefinita. Modificheremo COSTRUZIONE-PROVA cambiando la numerazione del passo 3(b)(v) in 3(b)(viii) e inserendo i passi seguenti:

3(b)(v) IF l'obiettivo è del tipo $\forall x(...x...)$
THEN sostituire il primo 'Derivare $\forall x(...x...)$' dell'agenda con:
Derivare $(...b...)$
Applicare ∀INTR $[(...b...)]$
dove '$(...b...)$' si ottiene dalla formula precedente cancellando il quantificatore universale più esterno e sostituendo con una costante individuale 'b' tutte le occorrenze ora libere della variabile quantificata. 'b' deve essere la prima costante in ordine alfabetico che non compare precedentemente in una riga accessibile.

3(b)(vi) IF l'obiettivo è del tipo $\exists x(...x...)$
THEN sostituire il primo 'Derivare $\exists x(...x...)$' dell'agenda con:
Derivare $(...b...)$
Applicare ∃INTR $[(...b...)]$
dove b è una costante individuale qualsiasi.

3(b)(vii) IF l'obiettivo è del tipo $(...b...)$
THEN sostituire il primo 'Derivare $(...b...)$' dell'agenda con:
Derivare $\forall x(...x...)$
Applicare ∀ELIM$[\forall x(...x...)]$
dove 'x' è una variabile.

Non è necessario apportare nessun'altra modifica a COSTRUZIONE-PROVA.

14.5 CONCLUSIONI

In questo capitolo abbiamo discusso la decidibilità della logica predicativa, ossia il problema dell'esistenza di un algoritmo che decida se un'argomentazione è valida o no.
Il teorema di Church afferma che la logica predicativa non ristretta non è decidibile. Tuttavia, alcune parti della logica predicativa sono decidibili, e abbiamo discusso la relazione fra questo fatto e il concetto di modello da noi introdotto. La tesi di Church, invece, è l'affermazione, non dimostrabile ma generalmente accettata, secondo cui i calcolatori sono in grado di eseguire solo ciò che è descritto da algoritmi.
Abbiamo infine presentato due metodi per la dimostrazione automatica di teoremi in logica predicativa: la risoluzione, descritta sia come regola di inferenza, sia come algoritmo, e un'estensione del metodo COSTRUZIONE-PROVA del Capitolo 11.

Applicazioni della logica enunciativa al progetto di circuiti logici e aritmetici

Nel Capitolo 2 avevamo detto che c'è una stretta relazione fra i valori di verità (TRUE e FALSE o 1 e 0) e la presenza o l'assenza di un flusso di corrente elettrica attraverso un circuito. Nell'ambito della progettazione fisica di un calcolatore, in realtà, un circuito è considerato come un dispositivo elettronico dotato di ingressi e di uscite, caratterizzati da un *livello di tensione*. Poiché l'uscita è completamente determinata dall'ingresso e dalla struttura del circuito, un circuito può essere considerato come una funzione. Se, in più, ammettiamo che ingressi e uscite possano assumere due soli valori (livello di tensione *alto* o *basso*, che rappresentano rispettivamente 1 e 0), possiamo considerare un circuito come una funzione di verità.

A.1 CIRCUITI ELETTRICI

I circuiti attualmente usati nei calcolatori di solito sono incisi su "chip" di silicio, ma, come avveniva nei primi calcolatori, possono essere costruiti con conduttori e componenti elettrici come resistori, condensatori, transistor e diodi. Non è comunque necessario sapere che cosa siano o come funzionino questi componenti; basta sapere che sono aggeggi che si possono combinare per formare circuiti.
Si consideri un circuito in cui l'ingresso è fornito da un generatore di tensione (ad esempio, una pila) controllato da un interruttore, e la cui uscita è un dispositivo elettrico (ad esempio una lampadina), come illustrato in Figura A.1.
In Figura A.1 l'interruttore è aperto, perciò nella lampadina non passa corrente; se invece l'interruttore viene chiuso, la lampadina si accende. Il funzionamento di questo circuito è illustrato nella tabella seguente:

Ingresso (interruttore)	**Uscita (lampadina)**
chiuso	accesa
aperto	spenta

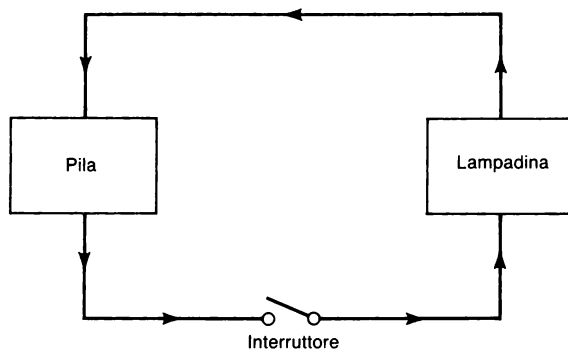



Figura A.1 Un semplice circuito (la freccia indica la direzione del flusso di corrente)

Consideriamo ora un circuito simile, ma con un *invertitore* al posto dell'interruttore. L'invertitore è a sua volta un circuito, con i propri fili e i propri componenti elettrici (di solito un resistore e un transistor); non ci interessa però la sua struttura, ma solo il suo comportamento: quando il generatore di tensione è in funzione, l'invertitore si comporta come un interruttore aperto e non lascia passare corrente verso la lampadina; quando invece il generatore è spento, l'invertitore funziona come un generatore di corrente e invia corrente alla lampadina. Il circuito si comporta quindi come una luce di emergenza, dotata di un generatore autonomo, che entra in funzione solo quando viene a mancare l'energia elettrica fornita dal generatore principale. Questo nuovo circuito contenente l'invertitore è rappresentato in Figura A.2.

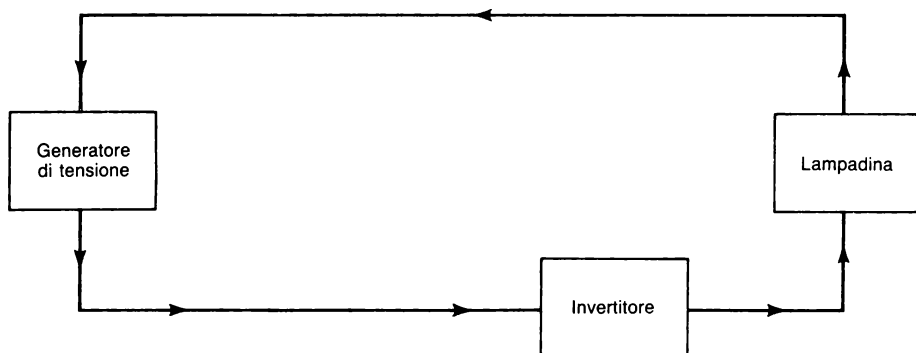



Figura A.2 Circuito con invertitore

Il funzionamento di questo circuito può essere sintetizzato con la seguente tabella:

Ingresso (generatore)	**Uscita (lampadina)**
spento	accesa
in funzione	spenta

Indicando la presenza di energia elettrica con 1 e la sua assenza con 0 questa tabella diventa:

Ingresso	**Uscita**
0	1
1	0

Ma questa non è altro che la tavola di verità di FNEG; dunque un invertitore si può usare per rappresentare la negazione. In effetti, l'invertitore è detto *porta NOT* (dall'inglese "not", che significa "non").

A.2 PORTE LOGICHE

Una *porta logica* è un circuito la cui tensione di uscita dipende dalle tensioni di ingresso secondo le leggi della logica enunciativa, delle quali può perciò costituire una rappresentazione. In teoria vi potrebbe essere un tipo di porta per ogni connettivo logico, ma in pratica le più comuni sono la porta NOT ("non": ¬), la porta AND ("e": ∧), la porta OR ("o": ∨), la porta NAND e la porta NOR. Come abbiamo visto, la porta NOT è un circuito il cui comportamento può essere rappresentato nel modo seguente:

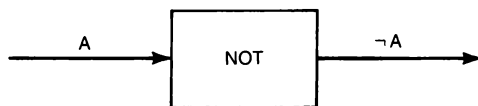



Figura A.3 Una porta NOT

Qui 'A' rappresenta la tensione d'ingresso; perciò è ragionevole rappresentare la tensione d'uscita con '¬A'. In ingegneria elettronica si usa un simbolo speciale per rappresentare la porta NOT:

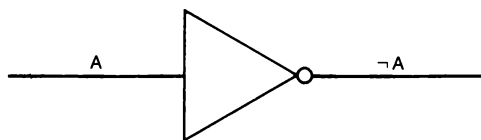



Figura A.4 Simbolo della porta NOT

Una porta AND a due ingressi (e, ovviamente, a una sola uscita) si può rappresentare così:

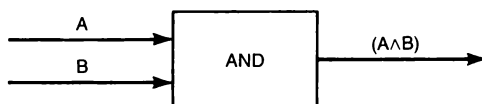



Figura A.5 Una porta AND

oppure così:

Figura A.6 Simbolo della porta AND

Se i livelli di tensione di ingresso sono entrambi bassi, il livello di tensione di uscita della porta AND è basso; se i livelli di tensione di ingresso sono uno alto e uno basso, il livello di tensione di uscita è basso; se infine i livelli di tensione di ingresso sono entrambi alti, il livello di tensione di uscita è alto.

ESERCIZI

1. Spiegate la relazione fra la porta AND e la tavola di verità della congiunzione, così come è stato fatto per la porta NOT e la negazione.

2. Descrivere il comportamento di una porta OR.

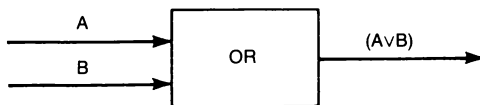



Figura A.7 Una porta OR

Figura A.8 Simbolo della porta OR

Spiegare la relazione fra essa e la disgiunzione.

3. Una porta NAND si può costruire con una porta AND e una porta NOT:

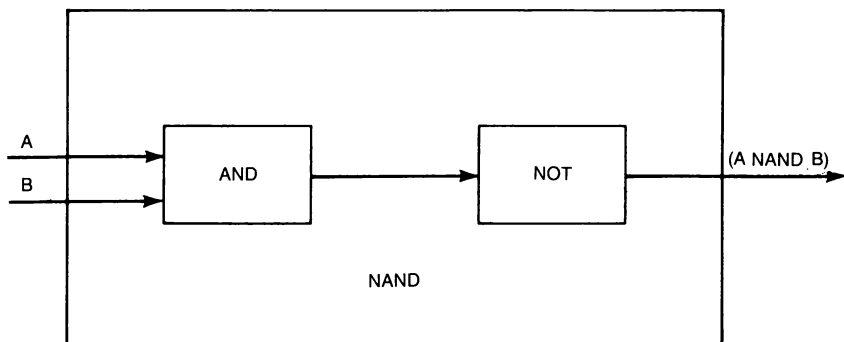



Figura A.9 Una porta NAND

a. Descriverne il comportamento.
b. Spiegare la relazione che la lega al connettivo logico NAND.
c. Spiegare la relazione fra la tautologia
   $((A \text{ NAND } B) \leftrightarrow \neg(A \wedge B))$
   e il comportamento di una porta NAND.

4. a. Mostrare come si costruisce una porta NOR.
   b. Descriverne il comportamento.
   c. Spiegare la relazione che la lega al connettivo logico NOR.
   d. Spiegare la relazione fra la tautologia
   $((A \text{ NOR } B) \leftrightarrow \neg(A \vee B))$
   e il comportamento di una porta NOR.
5. Fare lo stesso per la porta XOR (*Suggerimento:* si consideri la tautologia $((A \text{ XOR } B) \leftrightarrow ((A \vee B) \wedge \neg(A \wedge B)))$.

A.3 COMBINAZIONI DI PORTE LOGICHE

Queste porte logiche possono essere combinate in modo da rappresentare qualsiasi formula verofunzionale (così come abbiamo combinato una porta NOT con una porta AND per formare una porta NAND, nell'Esercizio 3).
Ad esempio,

$((A \wedge B) \vee \neg C)$

può essere rappresentato dal circuito seguente:

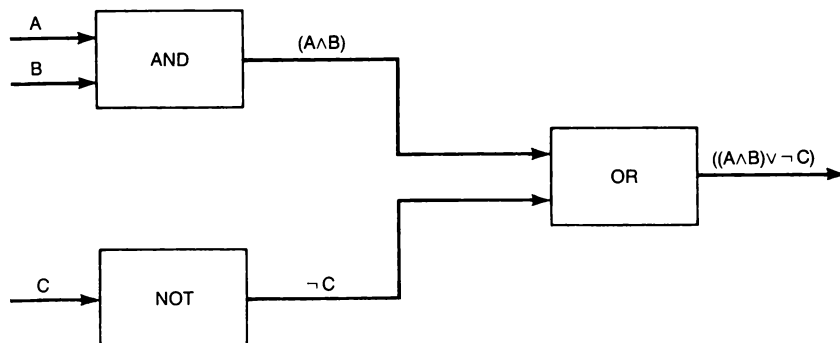



Figura A.10 Un circuito che rappresenta $((A \wedge B) \vee \neg C)$

Analogamente,

$(\neg A \vee \neg B)$

può essere rappresentato da

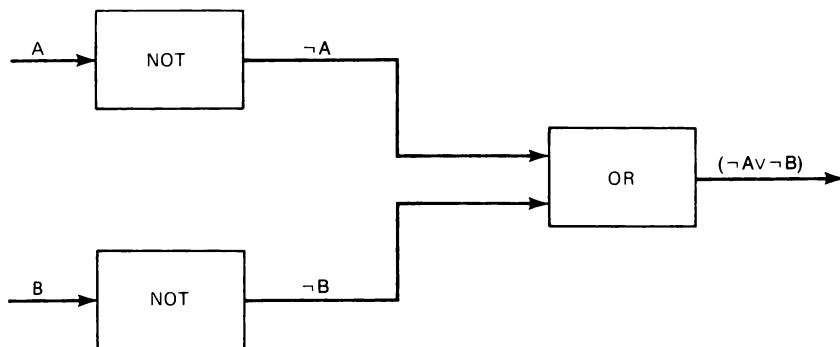



Figura A.11 Un circuito che rappresenta $(\neg A \vee \neg B)$

In virtù della legge di De Morgan (vedi Capitolo 7)

$$((\neg A \vee \neg B) \leftrightarrow \neg (A \wedge B))$$

esiste però un circuito costituito da un'altra combinazione di porte logiche che si comporta esattamente allo stesso modo:

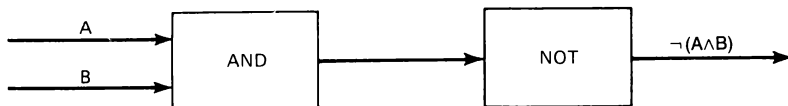



Figura A.12 Un circuito che rappresenta $\neg (A \wedge B)$ e quindi si comporta come il circuito di Figura A.11

Ovviamente, anche questo è un circuito NAND.

ESERCIZI

6. Usare l'altra legge di De Morgan per costruire un circuito che si comporti come una porta NOR.

7. Costruire i circuiti che rappresentano le formule seguenti:

   a. $((A \vee B) \vee C)$
   b. $(A \vee (B \vee C))$
   c. $((A \wedge B) \vee C)$
   d. $(A \wedge (B \vee C))$
   e. $(\neg A \vee B)$
   f. $\neg (A \wedge \neg B)$
   g. $(((\neg A \wedge \neg B) \vee C) \wedge \neg D)$
   h. $(\neg (A \wedge (\neg A \vee B)) \vee B)$

8. a. Costruire un circuito IF-THEN (*Suggerimento:* vedi Esercizio 7).
   b. Usando il circuito IF-THEN, costruire un circuito che rappresenti $((A \wedge (A \rightarrow B)) \rightarrow B)$

A.4 CONVERSIONE FRA ENUNCIATI LOGICI E CIRCUITI

Abbiamo visto che le informazioni relative ai valori di verità e alle funzioni di verità si possono descrivere in due modi: usando la logica enunciativa, oppure usando circuiti elettronici. Quando esistono due modi di rappresentare le stesse informazioni, è importante essere in grado di effettuare le conversioni fra le due rappresentazioni. Nel caso che stiamo esaminando, è importante essere sicuri che qualsiasi circuito logico avente una e una sola uscita si possa interpretare come un enunciato, e che ogni enunciato si possa rappresentare come un circuito logico con una uscita. Inoltre, se uno stesso circuito logico corrisponde a due enunciati diversi, questi devono essere logicamente equivalenti; allo stesso modo, se uno stesso enunciato può essere rappresentato da due circuiti diversi, questi devono presentare la stessa relazione fra ingresso e uscita.
In effetti, la relazione fra circuiti ed enunciati che è stata fin qui descritta garantisce che queste condizioni siano soddisfatte. Non dimostreremo questo fatto, ma il lettore, in base alle informazioni contenute in questo libro, dovrebbe essere in grado di svolgere da solo la dimostrazione. Nei paragrafi rimanenti vedremo alcuni esempi di conversione fra enunciati e circuiti.
Dato un circuito logico, il passaggio all'enunciato corrispondente è abbastanza diretto, anche se, ovviamente, quanto più il circuito è complicato, tanto più diventa noiosa la costruzione dell'enunciato corrispondente. Un circuito può anche avere più uscite; in tal caso può essere visto come una combinazione di circuiti separati, ciascuno dei quali ha una e una sola uscita. Si consideri il circuito di Figura A.13. Questo circuito ha due ingressi, A e B, e tre uscite, U_1, U_2 e U_3. Può essere visto come l'insieme di tre circuiti collegati fra loro: il primo è una porta NOR, i cui ingressi sono A e B e la cui uscita è U_1; il secondo è una porta OR i cui ingressi sono A e U_1 e la cui uscita è U_2; il terzo è un'altra porta OR, i cui ingressi sono B e U_1 e la cui uscita è U_3 .
Che enunciati rappresentano U_1, U_2 e U_3? Se le tensioni A e B rappresentano rispettivamente gli enunciati 'A' e 'B' , si ricava che:

1. U_1 rappresenta '(A NOR B)', o, equivalentemente, '$\neg(A \vee B)$'.
2. U_2 rappresenta '$(A \vee U_1)$', cioè, esprimendolo solo in funzione degli ingressi, '$(A \vee \neg(A \vee B))$'.
3. U_3 rappresenta '$(B \vee U_1)$', cioè '$(B \vee \neg(A \vee B))$'.

U_2 e U_3 possono però essere rappresentati in una forma molto più significativa. Se convertiamo U_2, cioè

$$(A \vee \neg(A \vee B))$$

in forma normale disgiuntiva (vedi Capitolo 7), otteniamo

$$(A \vee (\neg A \wedge \neg B))$$

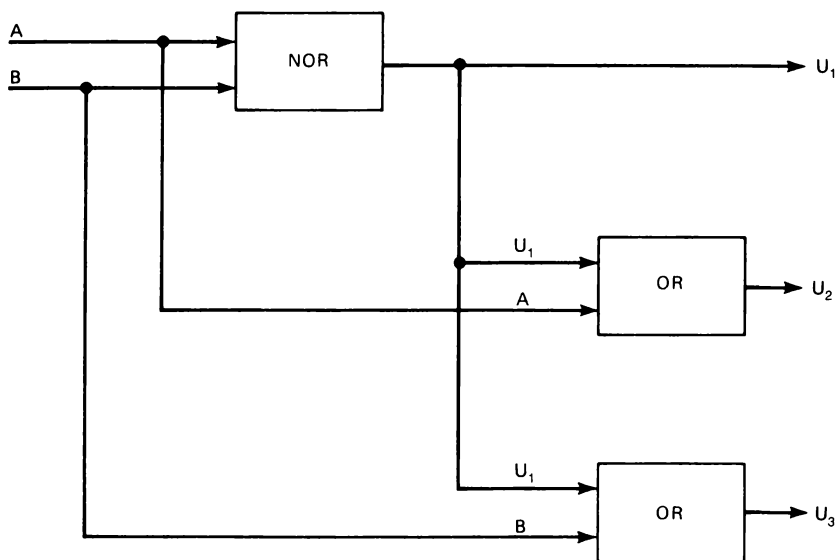



Figura A.13 Una combinazione di tre circuiti

che è logicamente equivalente a

$((A \vee \neg A) \wedge (A \vee \neg B))$

che, a sua volta, è logicamente equivalente a

$(A \vee \neg B)$

In modo analogo, si può dimostrare che U_3 rappresenta '$(B \vee \neg A)$'.
Si noti che gli enunciati corrispondenti alle uscite ricavati in modo diretto dal circuito non sono i più semplici. È vero anche il reciproco: se costruiamo un circuito a partire dai tre enunciati espressi nella forma più semplice, otteniamo il circuito di Figura A.14, che ha due porte NOT in più rispetto al primo circuito, e quindi non è il circuito più semplice che rappresenti questi enunciati.

A.5 SEMPLIFICAZIONE DEI CIRCUITI

L'insorgere di questo tipo di complicazioni nel caso di due rappresentazioni diverse di una stessa informazione è una caratteristica tipica della rappresentazione in generale. Una volta dimostrato che le rappresentazioni sono equivalenti, possiamo scegliere la più semplice, o la più adatta ai nostri scopi.

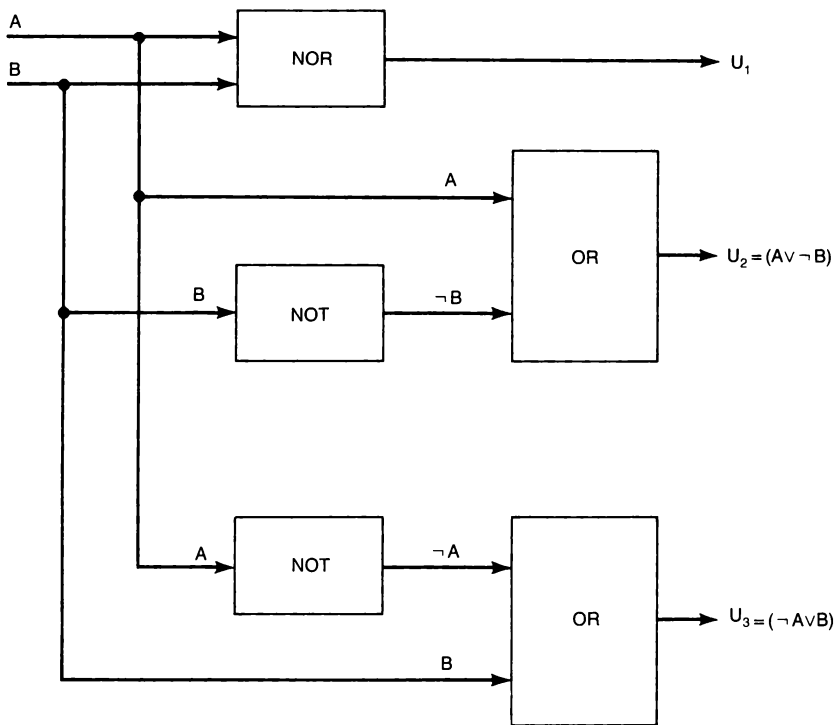



Figura A.14 Circuito avente lo stesso comportamento del circuito della Figura A.13

Se stiamo progettando un calcolatore preferiamo dunque il primo circuito al secondo, perché è indubbiamente meno costoso e più semplice da costruire e da riparare; se invece stiamo studiando le proprietà logiche dei circuiti, al primo insieme di enunciati preferiremo il secondo.
Esistono dei criteri per valutare queste scelte; ad esempio, si può dire che un circuito è più semplice di un altro se ha meno porte e, eventualmente, meno ingressi, cioè, in termini di logica enunciativa, meno connettivi e meno occorrenze di lettere proposizionali.
Come abbiamo visto nel Capitolo 8, tutti gli enunciati privi di quantificatori si possono esprimere usando solo $\neg$ e $\wedge$; perciò un progettista potrebbe cavarsela usando solo porte NOT e AND. In effetti, si possono costruire circuiti ancora più semplici, se si usano porte NAND e NOR ogni volta che è possibile, perché sono più facili da costruire rispetto alle sequenze di porte AND, OR e NOT. Le porte NAND e NOR sono particolarmente utili, perché, come si è visto nell'Esercizio

F del Capitolo 8, usando NAND (oppure NOR) come unico connettivo si possono esprimere tutti gli enunciati.
Esistono degli algoritmi di conversione fra enunciati e circuiti, che si basano sulle *mappe di Karnaugh*, una rappresentazione alternativa delle tavole di verità. Per un approfondimento, vedi Ennes, 1978; Kasper e Feller, 1983; oppure Malvino, 1976).

A.6 CIRCUITI ADDIZIONATORI

In questo paragrafo vedremo come si costruisce un circuito logico per sommare due numeri. Ciò dovrebbe far capire che non si esagera se si afferma che i calcolatori sono essenzialmente basati sulla logica.
Procediamo dal particolare al generale. Si consideri il seguente circuito:

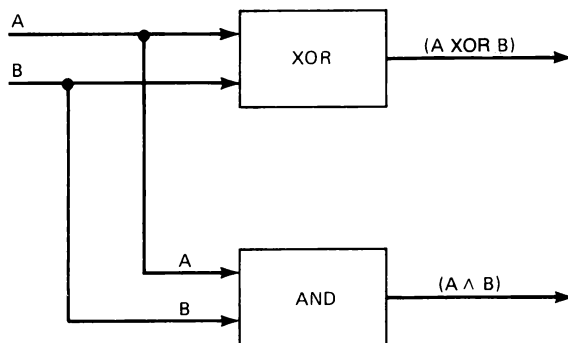



Figura A.15 Un semiaddizionatore

Una porta XOR e una porta AND ricevono in ingresso due tensioni A e B, ciascuna delle quali può essere alta o bassa. Rappresentando con 1 il livello alto di tensione e con 0 quello basso, possiamo tabulare le uscite di questo circuito nel modo seguente:

A	B	(A∧B)	(A XOR B)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Si consideri ora la seguente porzione di tabella per l'addizione binaria:

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	10

Se la somma di due numeri di un bit viene rappresentata come un numero di due bit (di cui il primo eventualmente uguale a zero), questa tabella diventa:

A	B	A + B
0	0	00
0	1	01
1	0	01
1	1	10

Ecco allora il legame fra la realizzazione elettronica di un circuito logico e l'operazione matematica di addizione: se le tensioni d'ingresso A e B sono interpretate come cifre binarie, il bit della colonna di destra della loro somma è rappresentato dall'uscita della porta XOR, mentre il bit della colonna di sinistra della loro somma è rappresentato dall'uscita della porta AND. Questo circuito è detto *semiaddizionatore.*

ESERCIZI

9. Un *addizionatore completo* è un circuito che realizza la somma di tre numeri binari di un bit. Date cioè tre cifre binarie A, B e C, l'addizionatore completo fornisce in uscita A + B + C. Un circuito sommatore completo è quello di Figura A.16.
   Spiegare come funziona.

10. Un *addizionatore binario* è un circuito che permette di sommare due numeri binari qualsiasi.
    In particolare se il numero binario A è una stringa di 4 bit $A_1A_2A_3A_4$ e il numero binario B è $B_1B_2B_3B_4$, la loro somma si può rappresentare così:

$$\begin{array}{r} A_1A_2A_3A_4 \\ +\,B_1B_2B_3B_4 \\ \hline S_0S_1S_2S_3S_4 \end{array}$$

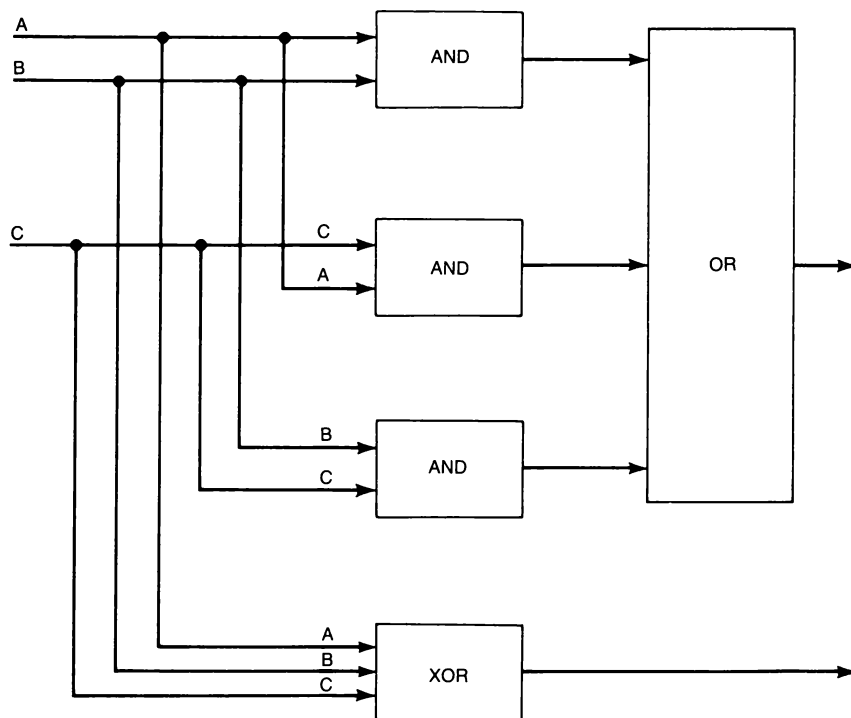



Figura A.16 Un addizionatore completo

Ad esempio, se A = 1010 e B = 1011, si ha:

$$\begin{array}{r} 1010 \\ +1011 \\ \hline 10101 \end{array}$$

dove $S_4 = 0 + 1 = 1$, $S_3 = 1 + 1 = 0$ con riporto di 1, $S_2 = 0 + 0 + 1$ di riporto $= 1$, $S_1 = 1 + 1 = 0$ con riporto di 1, $S_0 = 1$ del riporto. Un addizionatore per numeri binari di quattro bit è quello di Figura A.17.

a. Spiegare come funziona.
b. Spiegare come lo si usa per sommare:
   (i) 1000 e 0111
   (ii) 1111 e 1111

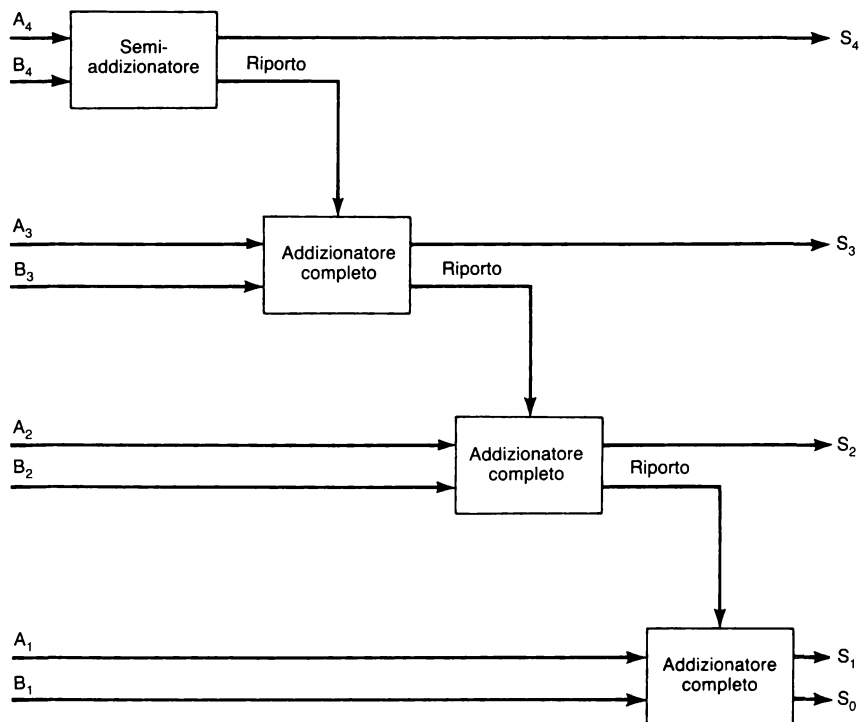



Figura A.17 Un addizionatore binario

B

Macchine di Turing

In tutto questo libro abbiamo fatto assegnamento in modo sostanziale sul concetto di algoritmo, definito nel Capitolo 2 come una sequenza dettagliata e finita di istruzioni, non ambigue ed eseguibili automaticamente, per lo svolgimento di un compito. Questo concetto è però espresso in modo intuitivo e informale: quanto deve essere dettagliato un algoritmo? Che tipi di istruzione sono permessi? Che cosa significa "automatico"? Come si possono evitare le ambiguità? Questa appendice vuole precisare il concetto di algoritmo, chiarendo il concetto di procedura automatica. La descrizione delle macchine di Turing presentata in questa appendice si basa su Clark e Cowell, 1976, pp. 44-49.

B.1 L'ANALISI DI TURING DELLA COMPUTAZIONE

Abbiamo detto che un algoritmo è una procedura automatica per svolgere un compito in un numero finito di passi. Se ci limitiamo ai compiti che si possono descrivere in qualche linguaggio, naturale o di programmazione, possiamo considerare tale procedura come una *computazione*. Nel 1937 il logico inglese Alan Turing (1912-1954) presentò un'analisi di questo concetto nell'importante articolo "On Computable Numbers, with an Application to the Entscheidungsproblem", che condusse, fra l'altro, al concetto di calcolatore a programma memorizzato. Il termine tedesco "Entscheidungsproblem" significa "problema di decisione" e indica il problema di determinare se una procedura per svolgere un dato compito produrrà una risposta affermativa o negativa in un numero finito di passi.
L'analisi di Turing comincia col considerare il concetto di esecuzione automatica di una computazione. Come minimo occorrono:

1. Un "calcolatore" (uomo o macchina) che effettui la computazione.
2. Della carta per appunti (un dispositivo di memorizzazione) su cui effettuare la computazione.

3. Un programma deterministico che il calcolatore deve eseguire, cioè una sequenza finita di istruzioni per effettuare la computazione manipolando i simboli sulla carta per appunti.

Questa descrizione piuttosto informale può essere raffinata. Supponiamo che la mente della persona, o del calcolatore, si possa trovare solo in un numero finito di *stati* distinti. Per comodità, supponiamo anche che la persona, o il calcolatore, abbia memorizzato il programma; in tal modo dobbiamo occuparci solo di due cose: il calcolatore e la memoria di lavoro.
Per quanto riguarda la memoria di lavoro, supponiamo di avere una grande quantità di carta: non necessariamente una quantità infinita, ma comunque sufficiente per permetterci di avere sempre dell'altra carta quando ci serve. Per poter descrivere con precisione ciò che è scritto sulla carta, immaginiamo che sia carta a quadretti e che ogni quadretto contenga un simbolo; è così possibile determinare in modo sistematico la posizione di ogni simbolo di ogni quadretto di ogni pagina della carta per appunti; ad esempio, si parte dal quadretto della prima riga e della prima colonna della prima pagina e si esaminano tutti i quadretti in sequenza, finché si trova il simbolo cercato oppure si raggiunge l'ultimo quadretto dell'ultima pagina senza averlo trovato.
Si immagini infine che, in ogni dato istante, la persona, o il calcolatore, che effettua la computazione possa vedere solo un numero finito e limitato di quadretti (o *caselle*) della carta per appunti.
Come successivo raffinamento faremo poi le seguenti ipotesi:

1. La persona, o il calcolatore, vede una sola casella della carta per appunti per volta, e ogni casella contiene al più un simbolo.
2. (a) La "carta per appunti" è un nastro continuo, suddiviso in caselle e potenzialmente infinito in entrambe le direzioni, nel senso che ad entrambi gli estremi è sempre possibile aggiungere un'ulteriore casella.
   (b) Ogni casella del nastro contiene inizialmente uno '0' o un '1' (i dati d'ingresso).

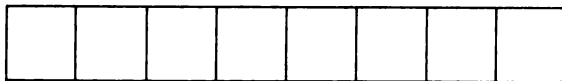



Figura B.1 Nastro di una macchina di Turing

3. Il programma memorizzato dalla persona, o dal calcolatore, è costituito non da istruzioni complesse (come "Trovare sul nastro la parola MOLTIPLICARE"), bensì da una sequenza finita di istruzioni dei cinque tipi seguenti:
   (a) START
   (b) IF si è nello stato P
       e si sta leggendo il simbolo S
         THEN

(i) Cambiare S in S′.
(ii) Passare nello stato Q.

(c) IF si è nello stato P
e si sta leggendo il simbolo S
THEN
(i) Spostarsi a destra di una casella.
(ii) Passare nello stato Q.

(d) IF si è nello stato P
e si sta leggendo il simbolo S
THEN
(i) Spostarsi a sinistra di una casella.
(ii) Passare nello stato Q.

(e) STOP.

Tra breve preciseremo ulteriormente queste istruzioni. Quello che così si ottiene è una *macchina di Turing*, che si può immaginare come una macchina fisica su ruote, con un'unità per la lettura e la scrittura sul nastro, detta *testina*, e un *registro* in cui è memorizzato lo *stato* in cui si trova la macchina.

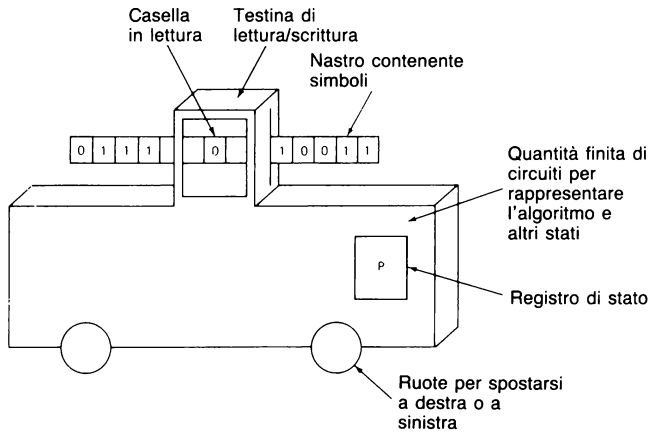



Figura B.2 Rappresentazione fisica di una macchina di Turing

B.2 MACCHINE DI TURING

Una *macchina di Turing* è costituita da una *configurazione istantanea* e da un programma le cui istruzioni sono tratte da un insieme specificato di *operazioni* e di *test*.
La configurazione istantanea, che è il corrispondente formale del nastro, è una coppia costituita da: (1) una stringa di '0' e '1', che rappresenta le informazioni

memorizzate sul nastro; (2) un numero intero positivo, che rappresenta il simbolo attualmente in lettura. Ad esempio,

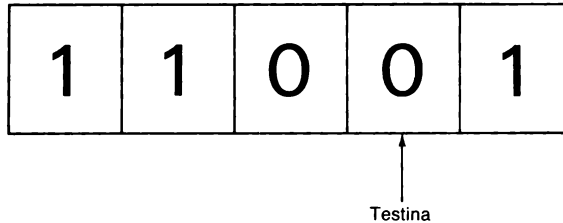



Figura B.3 Il nastro di una macchina di Turing rappresentato da (11001, 4) oppure da 110$\underline{0}$1

sarà rappresentato come

(11001, 4)

(dove il primo elemento della coppia è la stringa e il secondo è la casella in lettura), oppure come

110$\underline{0}$1

dove la sottolineatura rappresenta la posizione della testina. Il nastro *vuoto*, cioè privo di caselle, è rappresentato da una coppia il cui secondo elemento è 0. Indicheremo il nastro vuoto con il simbolo *e*.
Adotteremo un linguaggio di programmazione della macchina di Turing (LMT) costituito da sette operazioni e quattro test. Le operazioni sono:

1. START.
2. STAMPA-0. Cambia in 0 il simbolo attualmente in lettura. Più precisamente, STAMPA-0 cambia la configurazione istantanea *e* in (0, 1) e la configurazione istantanea $s_1s_2....s_{i-1}\, \underline{s}_i s_{i+1}...s_k$ in $s_1s_2...s_{i-1}\, \underline{0}s_{i+1}...s_k$. Ovviamente, se s_i è già 0, STAMPA-0 non modifica il nastro.
3. STAMPA-1. (Esercizio: descrivere l'effetto di STAMPA-1).
4. SINISTRA. Sposta la testina una casella più a sinistra, aggiungendo, se necessario, una nuova casella contenente uno 0. Si tenga presente che la nostra macchina di Turing è su ruote: è lei che si muove, non il nastro. Più precisamente, SINISTRA non cambia la configurazione istantanea *e*, mentre cambia $s_1...\underline{s}_i...s_k$ in $s_1...\underline{s}_{i-1}\, s_i...s_k$ e cambia $\underline{s}_1...s_k$ in $\underline{0}\, s_i...s_k$.
5. DESTRA. Questa operazione sposta la testina una casella più a destra, aggiungendo, se necessario, una nuova casella contenente uno 0. Più precisamente,

non cambia e, mentre cambia $s_i \ldots \underline{s}_i \ldots s_k$ in $s_1 \ldots s_i \underline{s}_{i-1} \ldots s_k$ e cambia $s_1 \ldots \underline{s}_k$ in $s_1 \ldots s_k \underline{0}$.

6. ELIMINA. Questa operazione non ha alcun effetto, a meno che la testina non si trovi all'estremità destra o sinistra del nastro, nel qual caso cancella il simbolo in lettura ed elimina dal nastro la corrispondente casella. In altre parole, ELIMINA lascia inalterate le configurazioni e e $s_1 \ldots \underline{s}_i \ldots s_k$, mentre cambia $\underline{s}_1 s_2 \ldots s_k$ in $\underline{s}_2 \ldots s_k$ e cambia $s_1 \ldots s_{k-1} \underline{s}_k$ in $s_1 \ldots \underline{s}_{k-1}$; inoltre fa sparire un nastro costituito da una sola casella, cioè cambia $\underline{s}$ in e.
7. STOP.

I test sono:

1. 0? e 1? verificano se il simbolo in lettura è 0 o 1. Più precisamente, se la configurazione è e il risultato di entrambi i test è FALSE, mentre se la configurazione è $s_1 \ldots \underline{s}_i \ldots s_k$ il risultato è TRUE se $s_i = s$ e FALSE altrimenti, ove s è 0 o 1, a seconda del test.
2. ESTREMO-SINISTRO? verifica se la testina si trova all'estremità sinistra del nastro. Se questa condizione è verificata, oppure se il nastro è vuoto, il risultato del test è TRUE, altrimenti è FALSE.
3. ESTREMO-DESTRO? verifica se la testina si trova all'estremità destra del nastro. Se questa condizione è verificata, oppure se il nastro è vuoto, il risultato del test è TRUE, altrimenti è FALSE.

B.3 PROGRAMMI PER LA MACCHINA DI TURING

Traduciamo alcuni semplici algoritmi in programmi in LMT.

NEGAZIONE

Cominciamo con l'algoritmo FNEG per il calcolo del valore di verità della negazione di un enunciato (vedi Capitolo 3). La prima decisione da prendere riguarda la rappresentazione dei valori di verità sul nastro. Una scelta ovvia consiste nel rappresentare un enunciato che vale TRUE con un nastro a una casella contenente un '1' e un enunciato che vale FALSE con un nastro a una casella contenente uno '0'. Dobbiamo poi decidere come dev'essere il risultato fornito in uscita. Si potrà ad esempio fare in modo che l'algoritmo si concluda con un nastro a due caselle, in cui il primo simbolo è il valore di verità originario, il secondo simbolo è la negazione di tale valore di verità e la testina è posizionata sul secondo simbolo. Si ricordi che è sempre possibile aggiungere nuove caselle al nastro. Dunque, se l'ingresso è '$\underline{1}$' l'uscita è '1$\underline{0}$', mentre se l'ingresso è '$\underline{0}$' l'uscita è '0$\underline{1}$'.

Si legge quindi il nastro d'ingresso: se contiene un '1', ci si sposta a destra e si scrive uno '0'; se contiene uno '0', ci si sposta a destra e si scrive un '1'.
In LMT rappresenteremo gli stati con M_0, M_1, M_2, M_3, eccetera. All'inizio (START) lo stato sarà sempre M_0. Il programma per calcolare FNEG nel modo appena descritto è:

1. START,
2. IF si è nello stato M_0 e si sta leggendo 0
   THEN
   (a) DESTRA.
   (b) Passare nello stato M_1.
3. IF si è nello stato M_0 e si sta leggendo 1
   THEN
   (a) DESTRA.
   (b) Passare nello stato M_2.
4. IF si è nello stato M_1 e si sta leggendo 0
   THEN
   (a) STAMPA-1.
   (b) Passare nello stato M_2.
5. IF si è nello stato M_2
   THEN STOP.

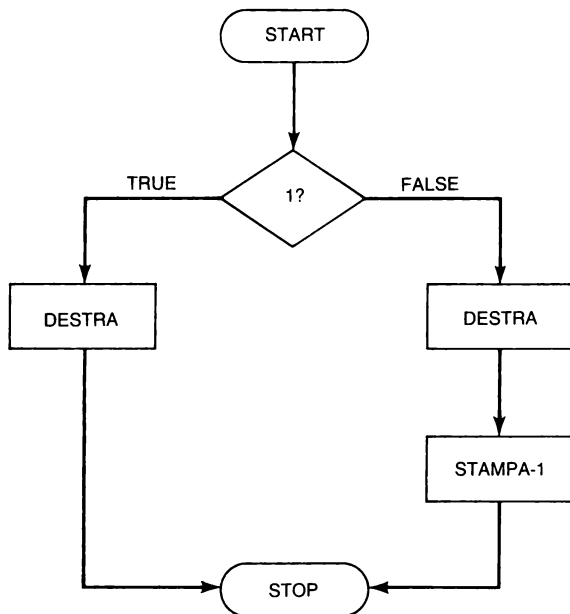



Figura B.4 Programma in LMT, sotto forma di diagramma di flusso, per il calcolo del valore di verità di una negazione

Si noti che è impossibile trovarsi nello stato M_1 mentre si sta leggendo 1; perciò questo caso non è contemplato dal programma.
È più comodo rappresentare il programma mediante un diagramma di flusso, ove lo stato è rappresentato dalla posizione entro il diagramma stesso (Figura B.4). La macchina di Turing comincia col leggere il nastro d'ingresso. Se contiene un '$\underline{1}$', la macchina di Turing si sposta a destra, aggiungendo automaticamente al nastro una casella contenente uno '0' e posta sotto la testina; la configurazione di uscita è perciò '1$\underline{0}$'. Se invece il nastro d'ingresso contiene uno '$\underline{0}$', la macchina di Turing si sposta a destra, aggiungendo una casella con uno '0', posta sotto la testina, poi stampa un '1' in tale casella; la configurazione di uscita è perciò '0$\underline{1}$'.

ESERCIZIO

Si supponga di rappresentare l'ingresso allo stesso modo e di rappresentare invece l'uscita nel modo seguente: se la configurazione d'ingresso è '$\underline{0}$', quella di uscita è '$\underline{1}$'; se la configurazione d'ingresso è '$\underline{1}$', quella di uscita è '$\underline{0}$'. In altre parole, sia il nastro d'ingresso, sia quello d'uscita hanno una sola casella, e quindi nel nastro d'uscita non è memorizzato il valore di verità originario. Scrivere un programma per la macchina di Turing, sotto forma di diagramma di flusso, che calcoli FNEG usando questa rappresentazione.

CONGIUNZIONE

Consideriamo ora un programma in LMT che realizzi FCNJ, cioè calcoli il valore di verità della congiunzione di due enunciati. Anche in questo caso dobbiamo decidere come rappresentare l'ingresso e l'uscita. Estendendo la rappresentazione usata prima, adottiamo le convenzioni seguenti: il nastro d'ingresso ha due caselle, ciascuna contenente o '0' o '1', che rappresentano i valori di verità dei due congiunti; la testina è inizialmente posizionata sulla prima casella, cioè sulla casella più a sinistra; il nastro di uscita ha tre caselle: le due originarie, immutate, e una nuova, posta sotto la testina e contenente il valore di verità della congiunzione. Ad esempio, se la configurazione d'ingresso è '$\underline{0}$1', quella di uscita sarà '01$\underline{0}$'. Si verifichi che il programma in LMT di Figura B.5 è conforme a queste specifiche.

ESERCIZI

Scrivere un programma in LMT, sotto forma di diagramma di flusso, per realizzare ciascuno dei seguenti algoritmi:

1. Un algoritmo per il calcolo di FCNJ, che cominci con il test 1? e usi la rappresentazione di ingresso e uscita sopra definita.

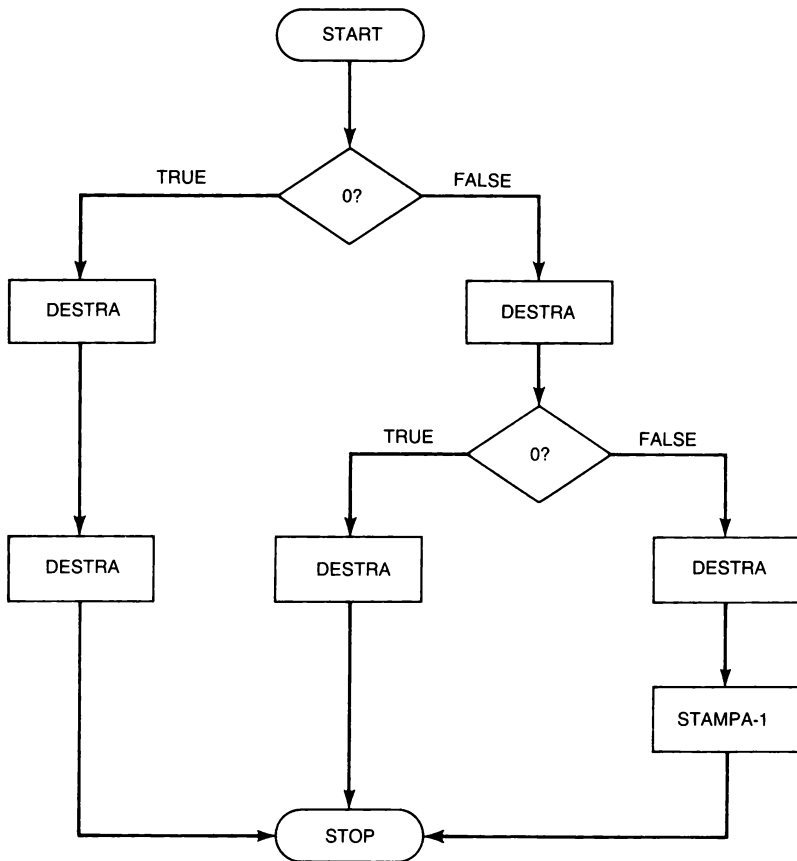



Figura B.5 Programma in LMT per il calcolo del valore di verità di una congiunzione

2. Un algoritmo per il calcolo di FCNJ che usi la rappresentazione "distruttiva" dell'uscita, in modo che, se la configurazione d'ingresso è, ad esempio, '0̲1', la configurazione di uscita sia '0̲'.
3. Un algoritmo per il calcolo di FDSJ. Specificare bene la rappresentazione dell'ingresso e dell'uscita.
4. Un algoritmo per il calcolo di FCND.
5. Un algoritmo per il calcolo di FBIC.
6. Un algoritmo per il calcolo di FXOR.
7. Un algoritmo per il calcolo di FNOR.

8. Un algoritmo per il calcolo di FNAND.
9. Un algoritmo per sommare due numeri interi positivi m e n. L'ingresso sia costituito da m '1', seguiti da uno '0' e da n '1'; adottando una notazione compatta, diremo che l'ingresso è

   1^m01^n

   L'uscita sarà costituita da $m + n$ '1', cioè sarà

   1^{m+n}

PALINDROMI

Come ultimo esempio di programmazione in LMT, abbozzeremo un programma che determina se il nastro d'ingresso contiene una *palindrome*, cioè una stringa di '0' e di '1' che si legga allo stesso modo da sinistra a destra e da destra a sinistra. Ad esempio,

000
010
001100

sono palindromi, mentre

01
110
0011110

non lo sono. Ovviamente, un nastro a una sola casella contiene sempre una palindrome. Sarà opportuno considerare come una palindrome anche il nastro vuoto. Il nastro d'ingresso conterrà la stringa da verificare; la testina sarà inizialmente posizionata sulla prima casella, cioè sulla casella più a sinistra. Il nastro di uscita avrà una sola casella, contenente un '1' se l'ingresso è una palindrome e uno '0' se non lo è. Vediamo due esempi:

Ingresso	**Uscita**
$\underline{1}01$	$\underline{1}$
$\underline{1}00$	$\underline{0}$

Adotteremo l'algoritmo seguente:

1. IF il nastro è vuoto
   THEN OUTPUT '1'. [perché è una palindrome]

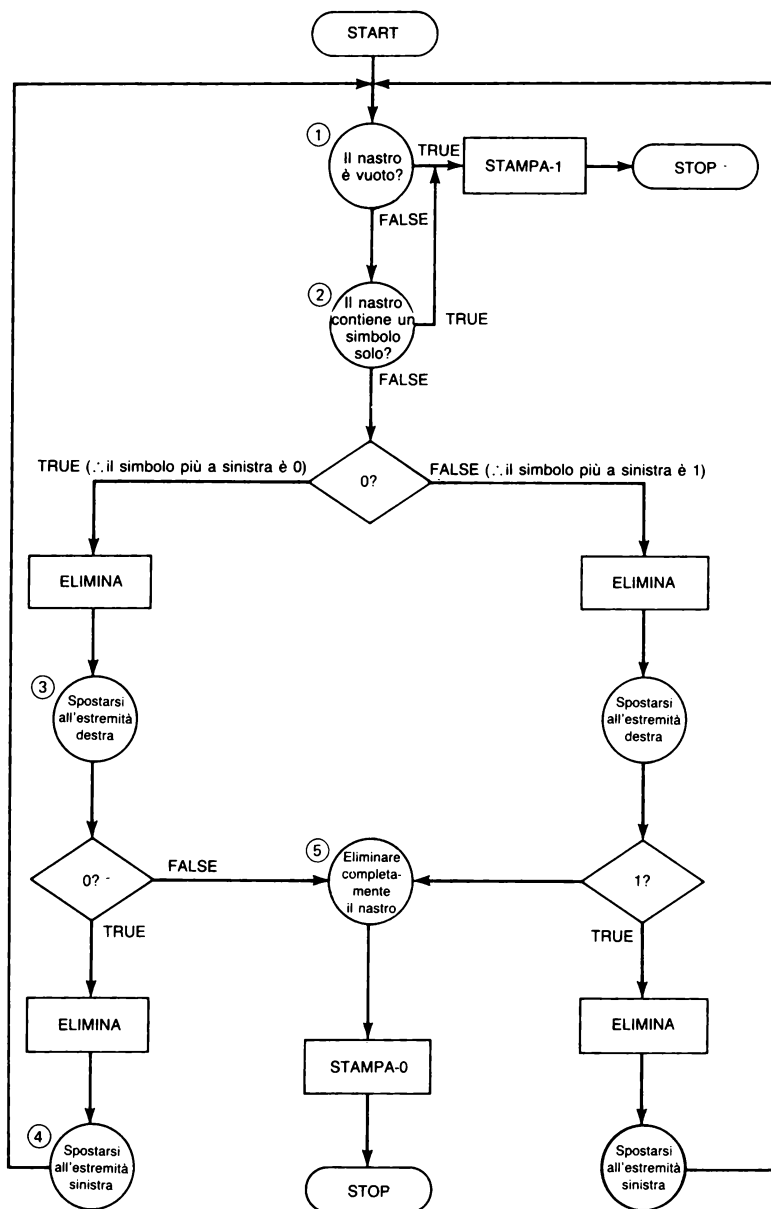



Figura B.6 Programma in LMT per il problema delle palindromi

2. IF il nastro ha una sola casella
   THEN OUTPUT '1'. [perché è una palindrome]
3. IF il nastro ha 2 o più caselle
   THEN
   (a) Confrontare il simbolo della casella più a sinistra con il simbolo della casella più a destra.
   (b) IF sono uguali
       THEN
       (i) Eliminare entrambi i simboli.
       (ii) GO TO passo 1.
   (c) IF non sono uguali
       THEN
       (i) Eliminare il nastro.
       (ii) OUTPUT '0' [perché non è una palindrome]
4. STOP.

Per rendersi conto che questo algoritmo funziona, lo si applichi ad alcuni esempi. Questa volta, per scrivere il programma, svolgeremo il progetto in modo discendente (dal generale al particolare), con raffinamenti per passi successivi; alcuni dettagli saranno poi lasciati agli esercizi. Introdurremo allora nei diagrammi di flusso un nuovo simbolo, il cerchio, per indicare i passi che devono essere raffinati ed espressi in termini delle operazioni e dei test elementari del LMT.
Il programma al livello più alto è quello di Figura B.6.
I cerchi etichettati da 1 a 5 rappresentano i sottoprogrammi che devono essere raffinati. Il raffinamento del sottoprogramma 1 è:

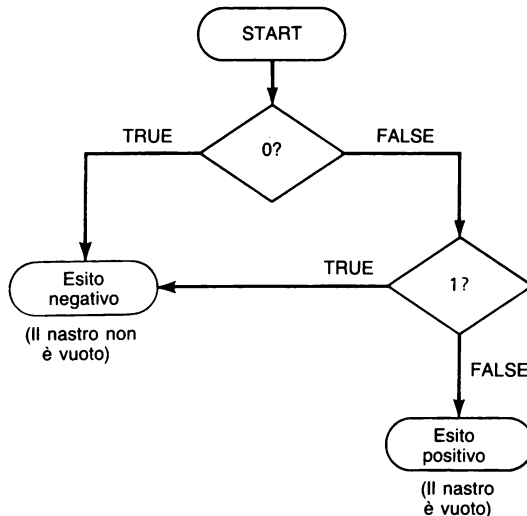



Figura B.7 Il sottoprogramma 1 del programma delle palindromi, espresso in LMT

Mentre il raffinamento del sottoprogramma 5 è:

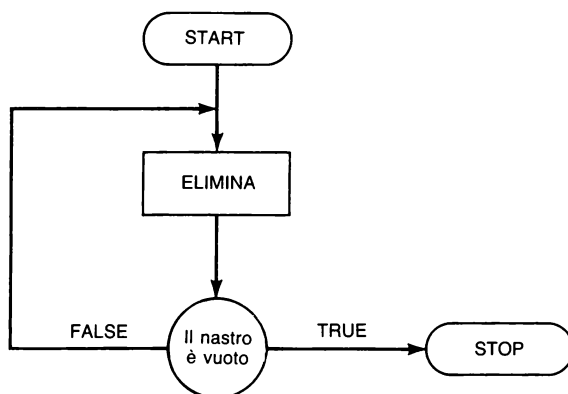



Figura B.8 Il sottoprogramma 5 del programma delle palindromi, espresso in LMT

ESERCIZI

10. Raffinare il sottoprogramma 2.
11. Raffinare il sottoprogramma 3.
12. Raffinare il sottoprogramma 4.
13. Raffinare il raffinamento del sottoprogramma 5 di Figura B.8.
14. Raffinare il sottoprogramma 5 senza usare il sottoprogramma 1 (*Suggerimento*: usare ESTREMO-SINISTRO?).

B.4 TESI DI CHURCH

Consideriamo la relazione fra una macchina di Turing e il nostro concetto di algoritmo.
Una versione della tesi di Church (vedi Capitolo 14), detta *tesi di Church-Turing*, afferma che il concetto intuitivo di computazione automatica può essere precisato definendo automatica una computazione effettuabile da una macchina di Turing. Ciò non può essere dimostrato, perché il concetto di computazione automatica è intuitivo e vago, mentre le dimostrazioni si fondano su concetti precisi; tuttavia esistono parecchi elementi a sostegno delle due affermazioni seguenti:

1. Ogni computazione che sembra intuitivamente automatica può essere effettuata da una macchina di Turing.
2. Ogni computazione effettuabile da una macchina di Turing è intuitivamente automatica.

L'elemento fondamentale a favore dell'affermazione 2 è l'analisi di Turing della computazione, che abbiamo discusso in un paragrafo precedente. Intuitivamente, sembra che ogni computazione effettuabile da una macchina di Turing sia intuitivamente automatica.
Abbiamo detto che la tesi di Church-Turing non può essere dimostrata; se però non è vera, può essere falsificata negando l'affermazione 1, cioè affermando che può esistere un procedimento intuitivamente automatico non eseguibile da una macchina di Turing, probabilmente perché le macchine di Turing, a causa delle dimensioni e delle operazioni limitate, non sono abbastanza potenti. Dopo tutto, le istruzioni sembrano un po' troppo semplici, soprattutto se si ammettono solo i simboli '1' e '0'.
Esistono però due tipi di motivi per credere all'affermazione 1. In primo luogo c'è una motivazione di origine empirica: tutti i procedimenti intuitivamente automatici finora ideati possono essere convertiti in programmi di una macchina di Turing.
In secondo luogo, tutte le altre teorie della computabilità si sono rivelate equivalenti alla teoria delle macchine di Turing.
Altre teorie della computabilità sono:

1. Il *lambda-calcolo* di Church, adottato nella formulazione originaria della tesi e usato in seguito come fondamento del linguaggio di programmazione LISP.
2. Gli *algoritmi di Markov*, che divennero poi il fondamento del linguaggio di programmazione SNOBOL.
3. La teoria delle *funzioni ricorsive parziali*, che sta alla base del *teorema dell'incompletezza di Gödel* e di buona parte della recente teoria della *programmazione strutturata*.
4. La teoria delle *equazioni ricorsive* di Herbrand-Gödel, che divenne poi il fondamento del linguaggio di programmazione ALGOL e, quindi, indirettamente, del linguaggio Pascal da esso derivato.
5. La teoria delle *macchine a registri*, che è il corrispondente matematico di un calcolatore digitale di von Neumann.

Bibliografia

Anderson, A.R. (ed.), *Minds and Machines*, Englewood Cliffs, N.J., Prentice-Hall, 1964

Boden, M., *Artificial Intelligence and Natural Man*, New York, Basic Books, 1977

Broad, C.D., *The Mind and Its Place in Nature*, London, Routledge, 1962

Church, A., *Introduction to Mathematical Logic*, Princeton, N.J., Princeton University Press, 1956

Clark, K.L., e D.F. Cowell, *Programs, Machines, and Computation*, London, McGraw-Hill, 1976

Dennett, D.C., *Brainstorms: Philosophical Essays on Mind and Psychology*, Montgomery, Vt., Bradford Books, 1978

Descartes, R., "Rules for the Direction of the Mind" (1628), in E.S. Haldane, e G.R.T. Ross (trad. ed ed.), *The Philosophical Works of Descartes*, Vol. I, London, Cambridge University Press, 1970, pp. 1-77

Dreyfus, H., *What Computers Can't Do: The Limits of Artificial Intelligence*, New York, Harper and Row, 1979

Ennes, H.E., *Boolean Algebra for Computer Logic*, Indianapolis, Sams, 1978

Gardner, M., *Logic Machines and Diagrams*, 2ª ed., Chicago, University of Chicago Press, 1982

Haugeland, J. (ed.), *Mind Design: Philosophy, Psychology, Artificial Intelligence*, Cambridge, Mass, M.I.T. Press, 1981

Hofstadter, D.R., *Gödel, Escher, Bach: An Eternal Golden Braid*, New York, Basic Books, 1979

Kasper, J., e S. Feller, *Digital Integrated Circuits: An Introduction for Students and Hobbyists*, Englewood Cliffs, N.J., Prentice-Hall, 1983

McCorduck, P., *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*, San Francisco, Freeman, 1979

Malvino, A.P., *Digital Computer Electronics*, New York, McGraw-Hill, 1983

Otto, H.R., *The Linguistic Basis of Logic Translation*, Washington, D.C., Uni-

versity Press of America, 1978

Raphael, B., *The Thinking Computer: Mind Inside Matter*, San Francisco, Freeman, 1976

Rich, E., *Intelligenza Artificiale*, Milano, McGraw-Hill, 1986

Slagle, J.R.: *Artificial Intelligence: The Heuristic Programming Approach*, New York, McGraw-Hill, 1971

Turing, A.M.: "On Computable Numbers, with an Application to the Entscheidungsproblem", *Proceedings of the London Mathematical Society*, 42 (1937), 230-265

———: "Computing Machinery and Intelligence" (1950), in Anderson (1964), pp. 4-30.

Weizenbaum, J., *Computer Power and Human Reason: From Judgment to Calculation*, San Francisco, Freeman, 1976

Winograd, T., *Language as a Cognitive Process*, Vol. I: *Syntax*, Reading, Mass, Addison-Wesley, 1983

Winston, P.H., *Artificial Intelligence*, Reading, Mass., Addison-Wesley, 1977

Indice analitico

La McGraw-Hill pubblica in tutto il mondo centinaia di libri di informatica per lo studio, la professione e il tempo libero. La produzione in lingua italiana comprende:

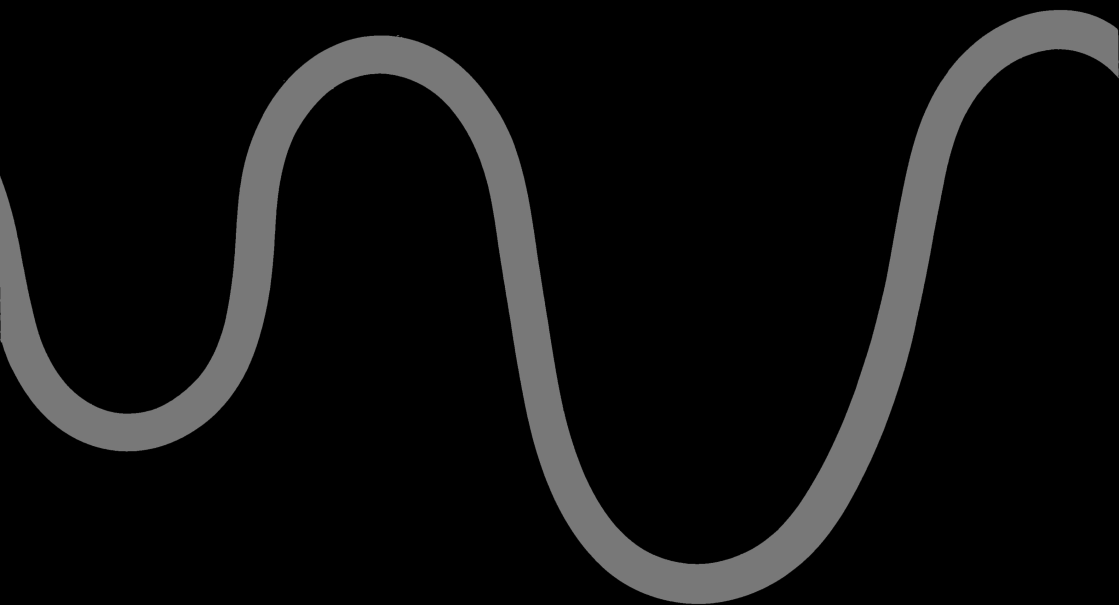
88 386 0001 5	J. Heilborn e R. Talbott, *Guida al Commodore 64*
88 7700 002 3	C.A. Street, *La gestione delle informazioni con lo ZX Spectrum*
88 7700 003 1	T. Woods, *L'Assembler per lo ZX Spectrum*
88 7700 004 X	R. Jeffries, G. Fisher e B. Sawyer, *Divertirsi giocando con il Commodore 64*
88 7700 005 8	G. Bishop, *Progetti hardware con lo ZX Spectrum*
88 7700 006 6	H. Mullish e D. Kruger, *Il BASIC Applesoft*
88 7700 007 4	N. Williams, *Progettazione di giochi d'avventura con lo ZX Spectrum*
88 386 0008 2	H. Peckham, *Il BASIC e il PC-IBM in pratica*
88 7700 009 0	H. Peckham, *Il BASIC e il Commodore 64 in pratica*
88 7700 010 4	S. Nicholls, *Tecniche avanzate in Assembler con lo ZX Spectrum*
88 7700 011 2	K. Skier, *L'Assembler per il Commodore 64 e il VIC-20*
88 7700 012 0	S. Kamins e M. Waite, *Programmazione umanizzata in Applesoft*
88 7700 013 9	A. Pennell, *Guida allo ZX Microdrive e all'Interface 1*
88 7700 015 5	P. Cohen, *Grafica e animazione con gli Apple II*
88 7700 016 3	C. Duff, *Guida al Macintosh*
88 7700 017 1	G. Kane, *Il manuale MC68000*
88 386 0018 X	P. Hoffman e T. Nicoloff, *Il manuale MS-DOS*
88 386 0019 8	E.M. Baras, *Come usare il Symphony*
88 7700 020 1	S. Nicholls, *Grafica avanzata con lo ZX Spectrum*
88 386 0021 X	L.J. Graham e T. Field, *Guida al PC-IBM*
88 7700 022 8	T. Field, *Come usare MacWrite e MacPaint*
88 7700 024 4	H. Peckham, *Il BASIC e gli Apple II in pratica*
88 7700 025 2	C. Morgan e M. Waite, *Il manuale 8086/8088*
88 7700 026 0	W. Ettlin, *Come usare il Multiplan*
88 7700 027 9	G. Mainis, *Il manuale ProDOS*
88 7700 028 7	J. Jones, *Il SuperBASIC del QL*
88 7700 029 5	C. Opie, *L'Assembler per il QL*
88 7700 030 9	W. Ettlin e G. Solberg, *Il BASIC Microsoft*
88 386 0031 7	D.L. Toppen, *Il Forth in pratica*
88 7700 032 5	R. Person, *Le meraviglie dell'animazione con gli Apple II*
88 386 0033 3	P.A. Sand, *Programmazione avanzata in Pascal*
88 7700 034 1	P. Hoffman, *Il manuale MSX*
88 7700 035 X	R. Person, *Le meraviglie dell'animazione con il PC-IBM*
88 7700 036 8	W. Ettlin, *Il Multiplan per il Macintosh*
88 386 0037 6	D. Kruglinski, *Introduzione al Framework*
88 386 0038 4	L. Barnes, *Come usare il dBase II*

88 386 0040 6	L. Barnes, *Come usare il dBase III*
88 386 0041 4	W. Ettlin e G. Solberg, *Il GW-BASIC per Personal Computer Olivetti*
88 386 0042 2	D. Watt, *Il LOGO per il Commodore 64*
88 386 0043 0	T.J. Byers, *Guida al PC AT IBM*
88 386 0044 9	D. Kater e R. Kater, *Guida alle stampanti Epson*
88 386 0045 7	R.L. Tokheim, *Progetti di circuiti elettronici per microcalcolatori*
88 386 0047 3	J. Heilborn, *Guida al Commodore 128*
88 386 0048 1	H. Peckham, *Programmazione strutturata in BASIC*
88 386 0052 X	P. Hoffman e T. Nicoloff, *Il manuale MS-DOS per Personal Computer Olivetti*
88 7700 601 3	S. Harrington, *Computer Graphics - Corso di programmazione*
88 7700 602 1	O. Lecarme e J.L. Nebut, *Pascal - Guida per programmatori*
88 386 0603 X	M. McGilton e R. Morgan, *Il sistema operativo UNIX*
88 386 0604 8	E. Rich, *Intelligenza Artificiale*
88 386 0605 6	M. Schagrin, J. Rapaport e R. Dipert, *Logica e computer*
88 386 0606 4	W. Newman e R. Sproull, *Principi di Computer Graphics* (in preparazione)
88 386 0607 2	L. Hancock e M. Krieger, *Il linguaggio C*

La produzione software comprende:

88 7700 902 0	C.A. Street, *PROFILE 2 - Foglio elettronico integrato per lo ZX Spectrum*
88 7700 903 9	S. Nicholls, *Routines in Assembler per la grafica avanzata con lo ZX Spectrum*
88 7700 904 7	A. Bleasby, *Assembler/Disassembler per il Commodore 64*
88 7700 905 5	ACS Software, *ZX Spectrum Monitor*
88 7700 906 3	G. Fitzgibbon, *Projector 1 - Uno strumento per costruire presentazioni grafiche con lo ZX Spectrum*
88 386 0907 1	C. Opie, *QL Machine Code Editor/Assembler*
88 386 0908 X	B. Thompson e W. Thompson, *Sistema Esperto McGraw-Hill per personal MS-DOS*
88 386 0909 8	A. Tal, *Generatore di lezioni per il Commodore 64*
88 386 0910 1	A. Tal, *Generatore di lezioni per gli Apple II* (in preparazione)
88 386 0911 X	B. Thompson e W. Thompson, *Sistema Esperto McGraw-Hill per Apple II*

Morton L. Schagrin
William J. Rapaport
Randall R. Dipert

Logica e computer

Lire 37 000
(IVA 2% inclusa)

ISBN 88 386 0605 6

SCHAGRIN
RAPAPORT
DIPERT

LOGICA E COMPUTER

McGraw-Hill